TECHNICAL BANK
アスキー・テクニカル・バンク
パソピアの内部構造
東 潤一／池田公平／浅野泰之 共著
アスキー出版局

③

SCREEN 2 COLOR SAMPLE

①

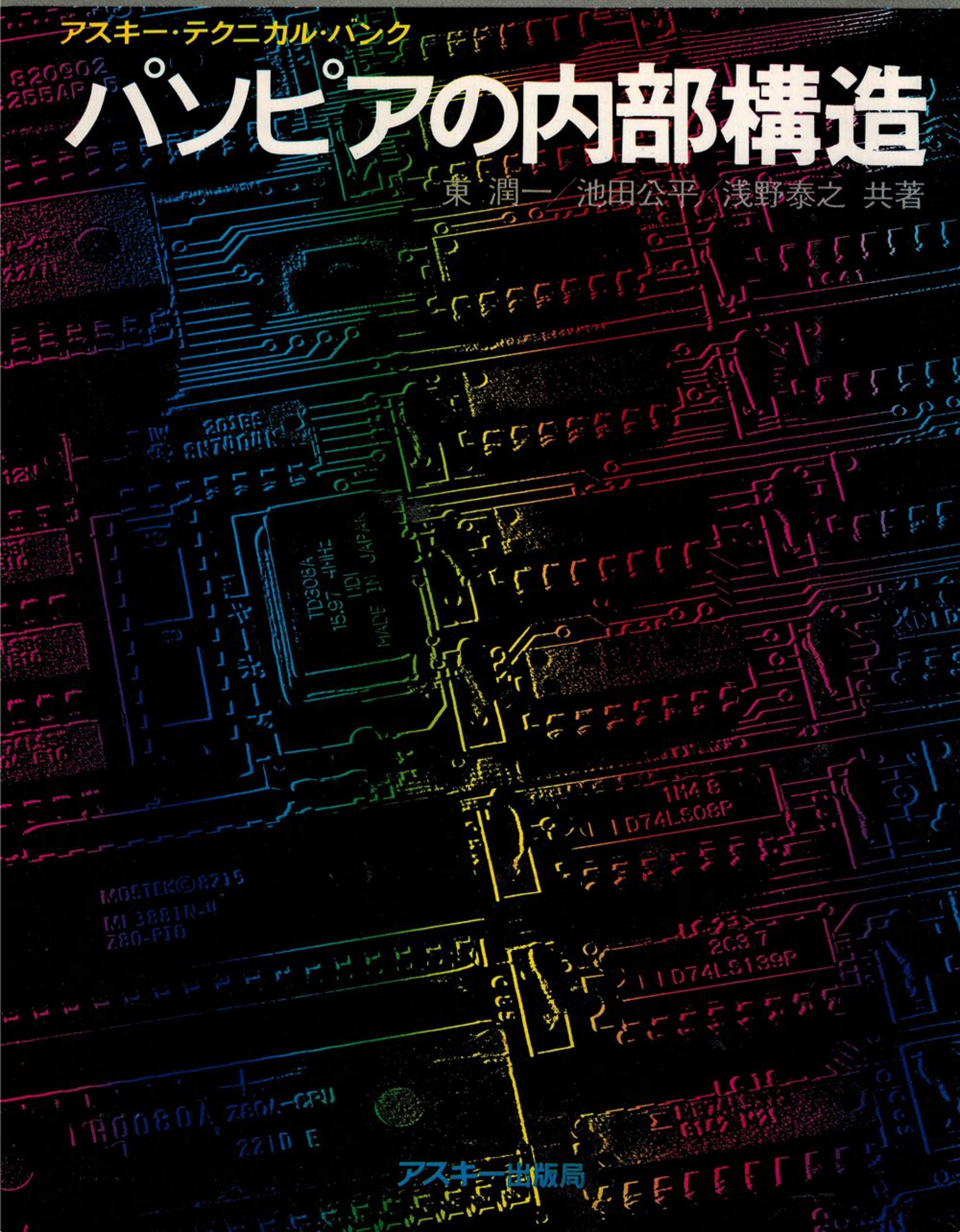
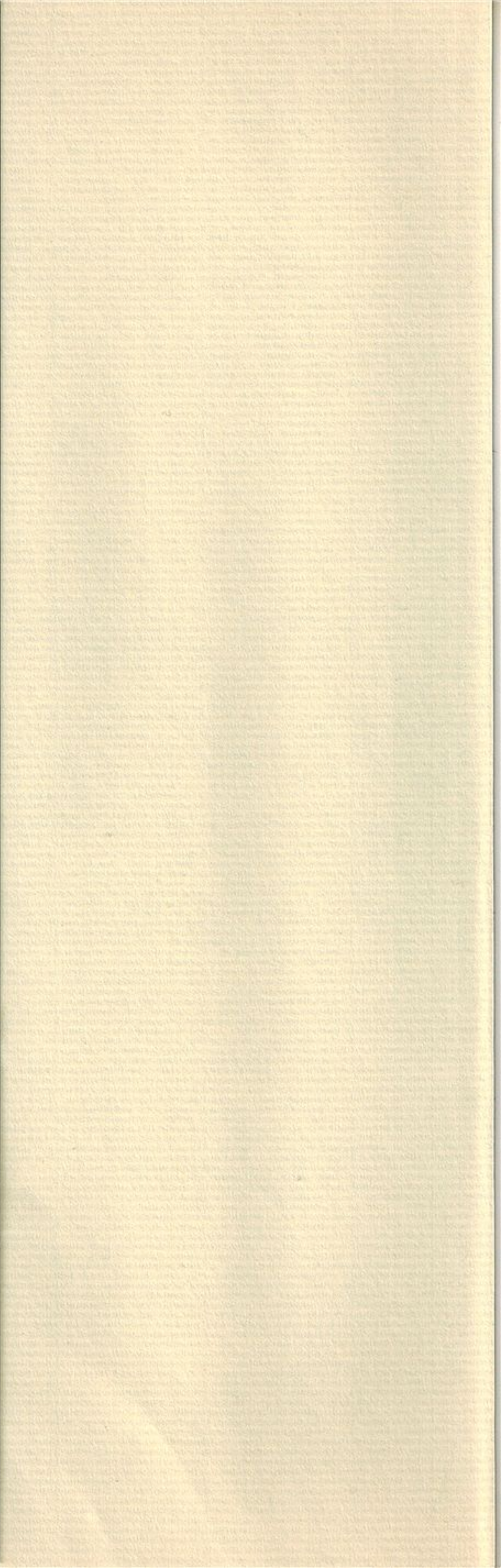
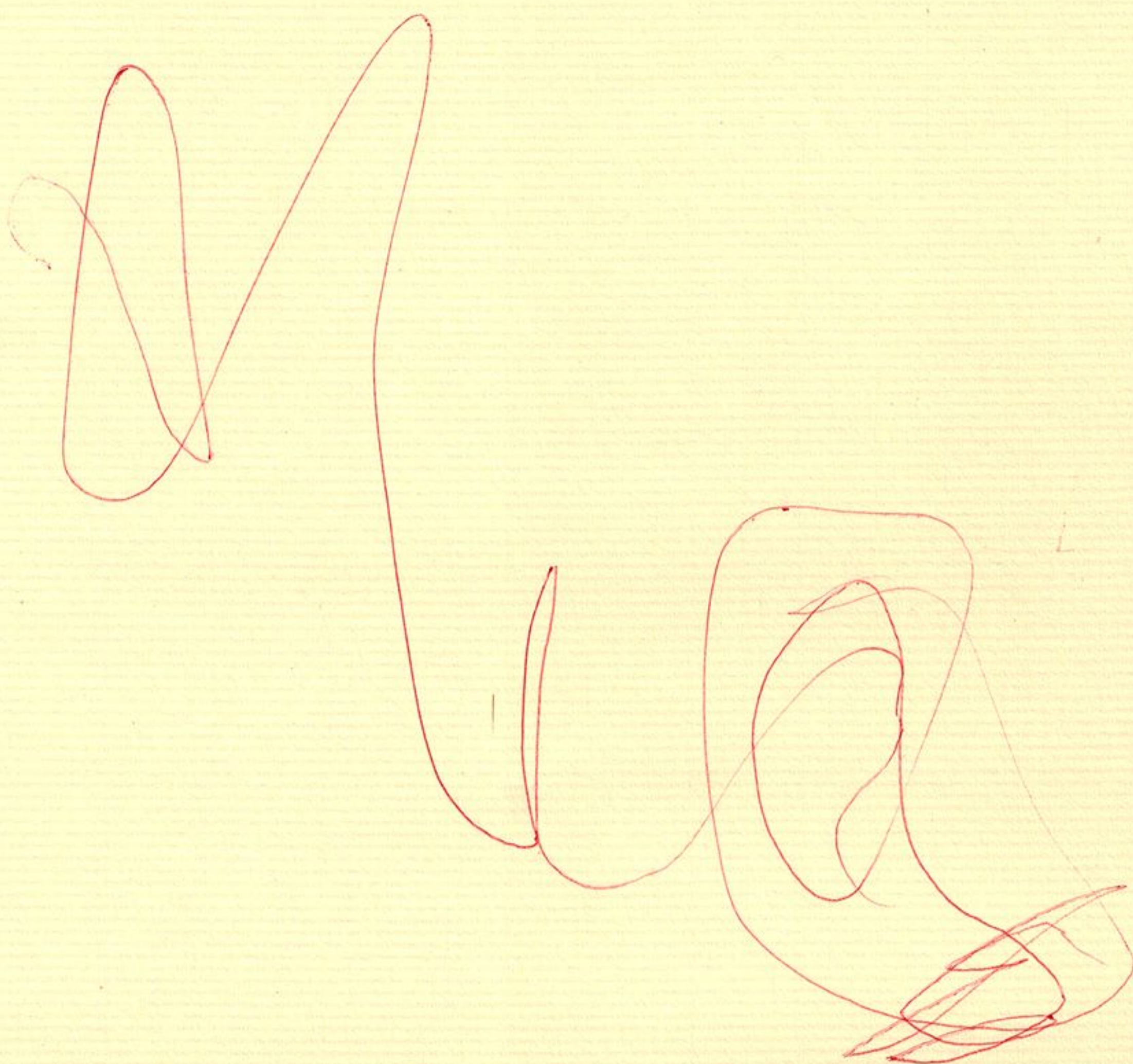
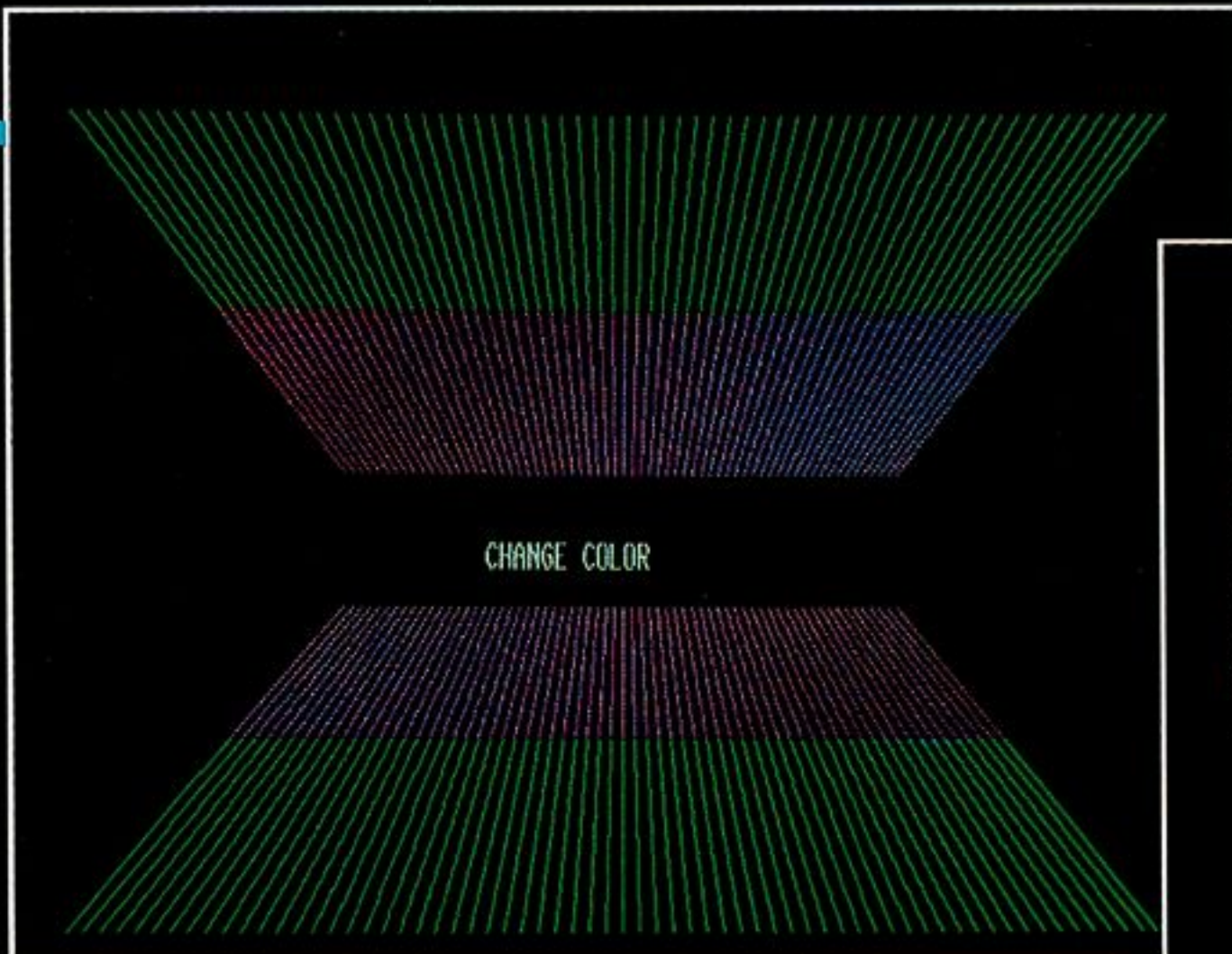
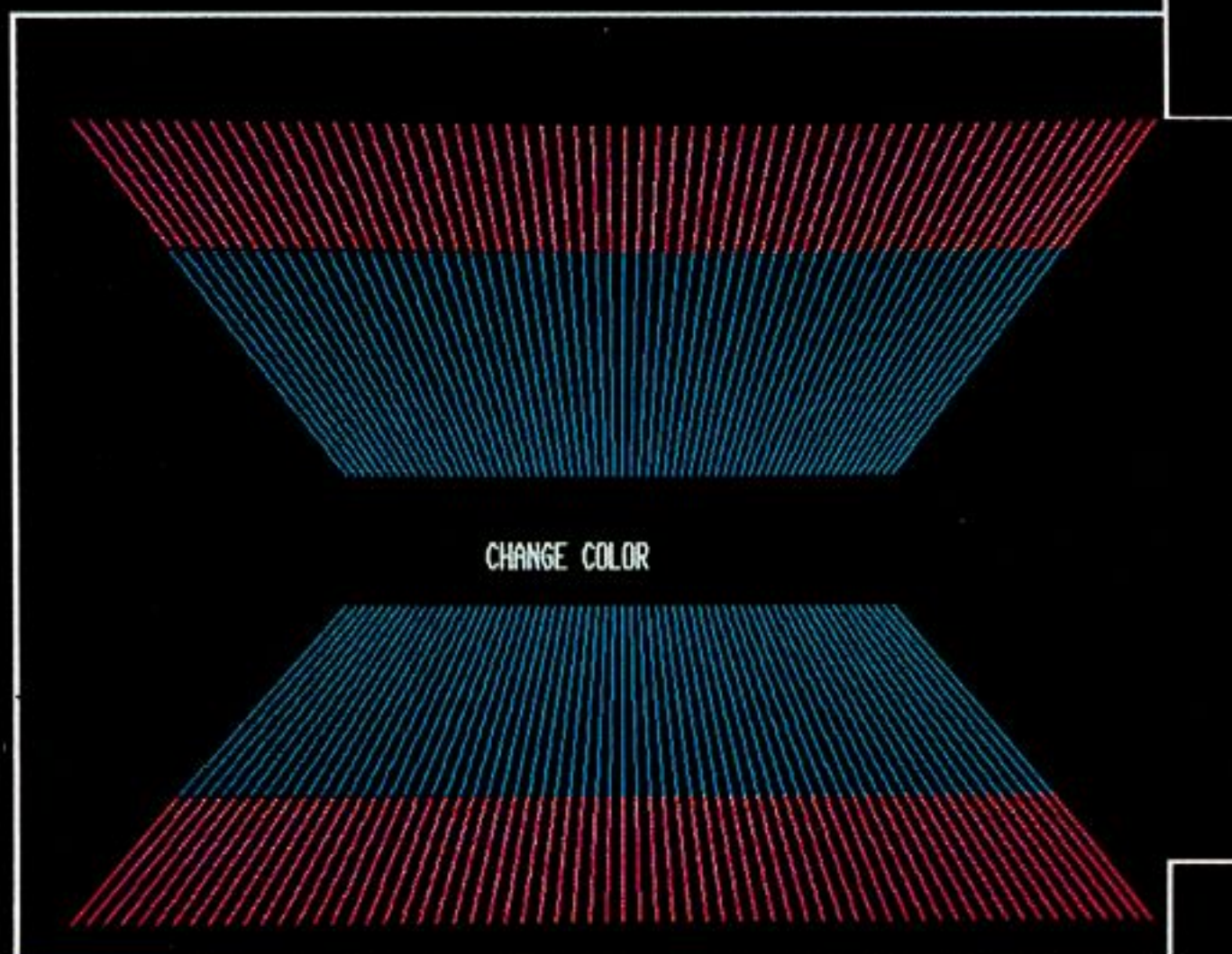
④

《 PASOPIA 》

ﾃﾞｰﾀ ｽｳ : 20
ﾍｲｷﾝ : 5.30183
Max : 9.84546
Min : .817514

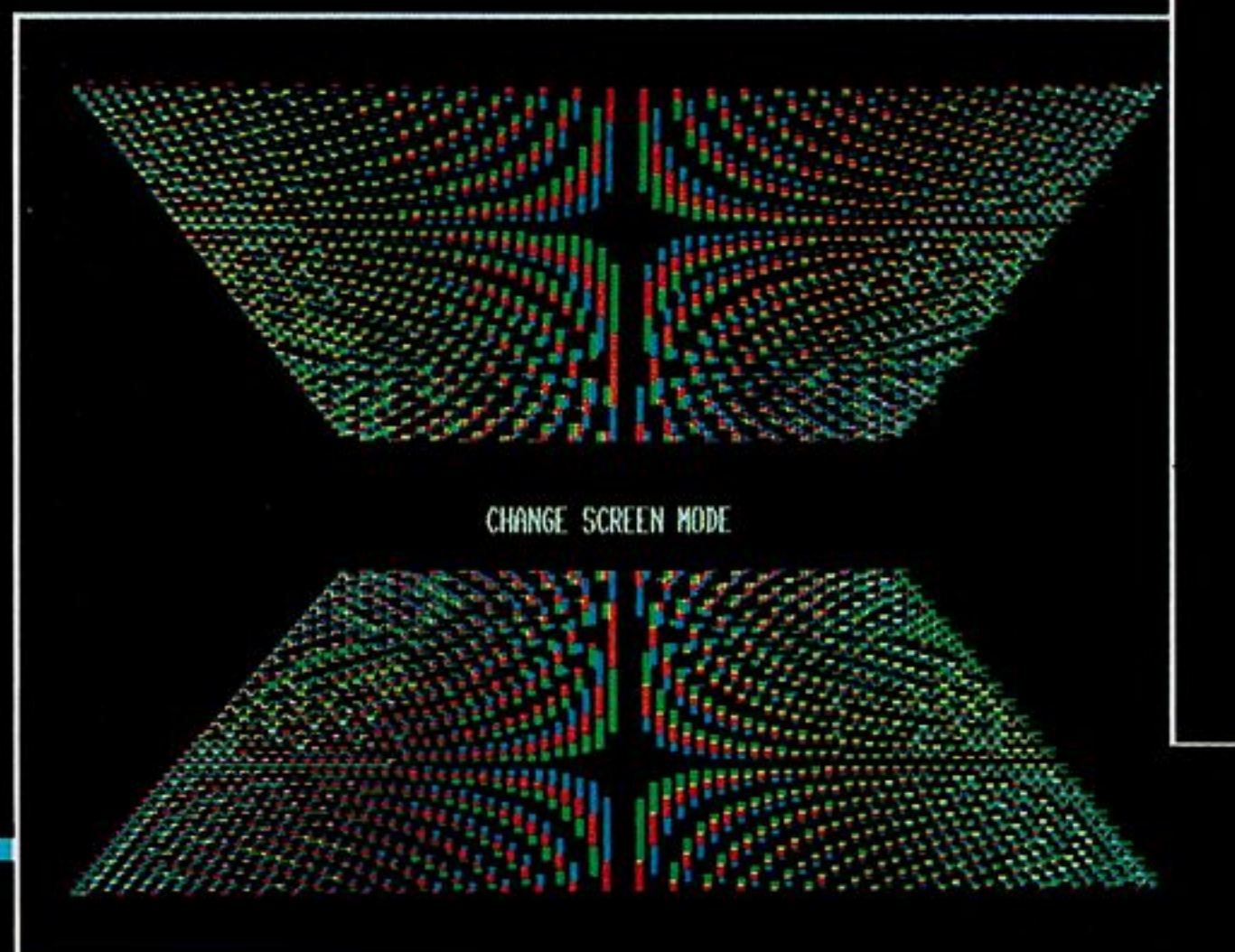
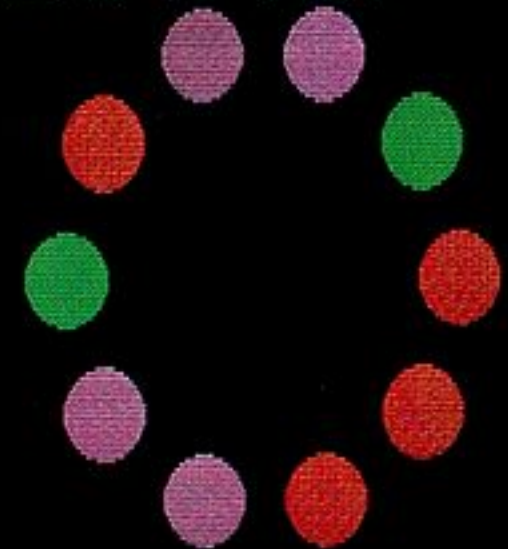
⑦

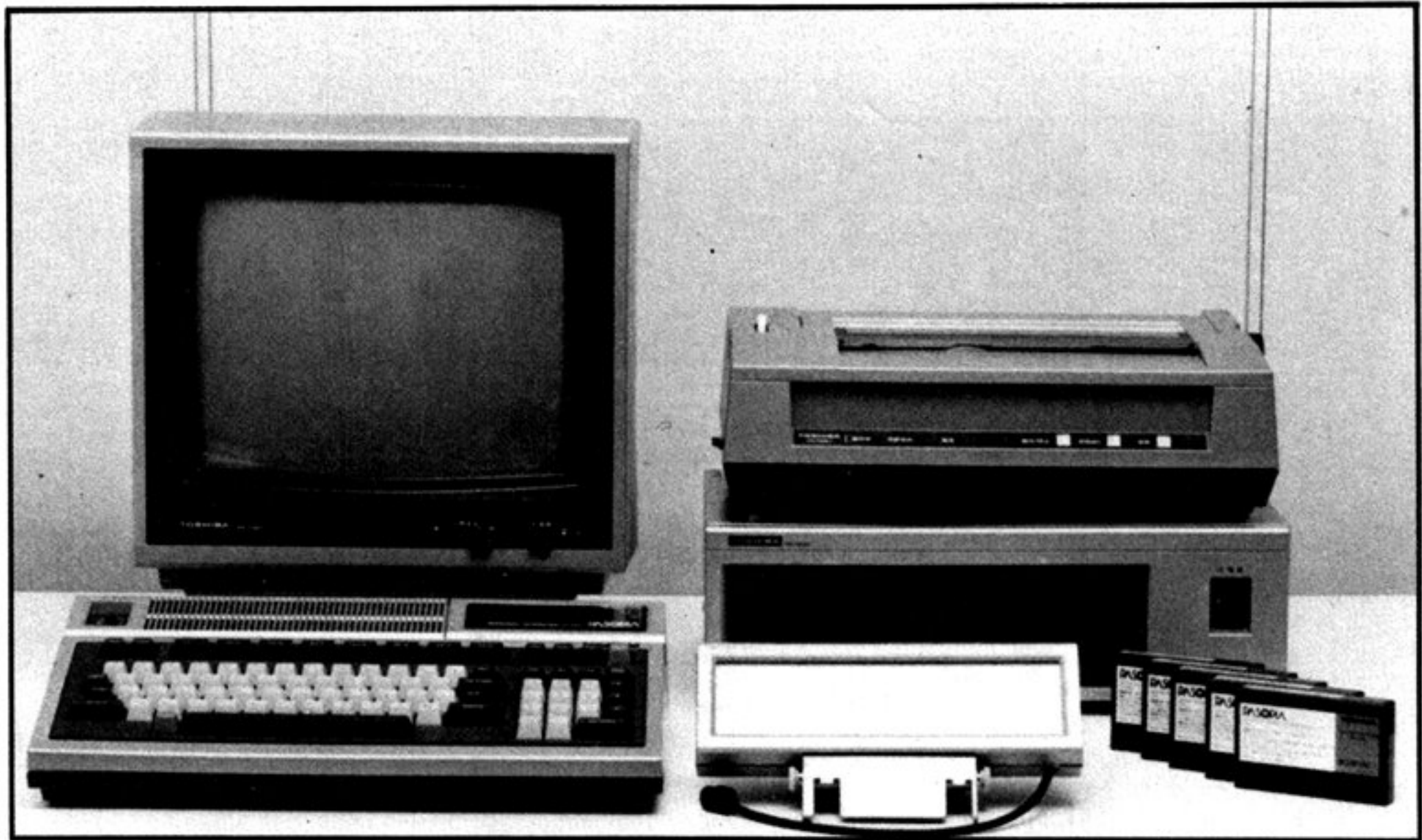
⑤

⑧

```
1000 ' ***
1010 ' ***   COLOR REMARK SAMPLE
1020 ' ***
1030 '       PASOPIA ﾊ ｺﾉﾖｳﾆ REM ﾆ ｲﾛ ｶﾞ ﾂｷﾏｽ!!!
1040 '
1050 ' This program is sample of 'color-remark'.
1060 ' If you want to color with remark , you must type next command
1070 '                                          in direct mode.
1080 '     FOR I=1 TO 7:KEY I,CHR$(248+I):NEXT
1090 ' Then programble function key has color-attribute-charactor.
1100 ' You type function key behind remark.
Ok.
-
```

⑥

⑨

②

⑩

パソピアの内部構造

東　潤一
池田公平　共著
浅野泰之

アスキー出版局

著者まえがき

東芝のパーソナル・コンピュータPASOPIAは，2つのBASIC(T-BASICとOA-BASIC)の採用など，ユーザーの立場に立った独自の方針を持つマシンとして，1981年秋の発表以来，高い評価を受けて来ました．さらに最近では，MINI-PASCAL，CP/Mの発売もあり，ますますその機能は広がっています．

しかし，その多彩な機能を引き出すための解説書ということになると，わずかに入門書的なものがあるだけで，とうてい十分とは言えず，PASOPIA本来の機能をフルに利用したい，というユーザーの希望を活かすことができませんでした．

BASICの内部構造はどうなっているのか，機械語を使うにはどうすればよいのか，ハードウェアの構成はどうなっているのか，といったさまざまな疑問をかかえているユーザーも少なくないと思います．

本書は，そのような方々のために，PASOPIAを徹底的に使いこなす手引きとして，マニュアル・雑誌記事などでは触れられていない情報を整理したものです．

ソフトウェアに関しては，T-BASIC(Ver.1.1)を中心に，OA-BASIC，MINI-PASCAL，CP/Mのすべてにわたって解説を加えており，ハードウェアに関しても，多くの図表・写真をまじえ，具体的に詳しく説明してあります．

紙数の制約もあり，書き足りない点もあるかと思いますが，PASOPIA初の本格的解説書として，必ずや，多くのユーザーの方々に満足していただける内容であると考えております．読者の方々が，本書によって，PASOPIAの持つ魅力をよりいっそう引き出していただけるよう願っています．

なお，本書の原稿執筆に関しては，ハードウェア関係を池田が，BASIC関係を東が，また，MINI-PASCAL，CP/Mを浅野が担当いたしました．付録に関してはI/Oポートを池田が，T-BASICの一覧表を浅野が担当しております．

本書の出版に関して，㈱東芝・オフィスオートメーション事業部の鈴木氏，山田氏に大変お世話になりました，紙面をかりて感謝の意を表します．

1983年2月
著者代表　東　潤一

この本を読む前に

本書は，東芝パーソナルコンピュータ・PASOPIA(パソピア)のユーザーの中で，

1.T-BASICは一通りマスターしたが，もう一歩つっこんだノウハウを知りたい，という方
2.ハードウェアに興味のある方
3.パソピア上で動くいろいろなシステム(パスカル・CP/M等)について興味のある方

を対象として書かれています．

初心の方には多少難しい内容もありますが，パソピアを十分活用するために必要なテクニックを満載しましたので，マニュアルと合わせてお読み下されば，理解が深まると思います．

本書で使用した主なシステムは，T-BASIC Ver.1.1(ROM版)，OA-BASIC Ver.D1.0(ディスク版)，MINI-PASCAL(ROMPAC版)です．その他のシステムについては，必要に応じて触れました．内部ポインタ値など，異なるものが多いので注意して下さい．

本書の構成を上げると，1章のハードウェアはパソピアのハード全般について，2章から6章まではT-BASIC Ver.1.1，7章はOA-BASIC，8章はMINI-PASCAL，9章はCP/M，について述べてあります．

各章の内容は以下の通りです．

第1章　ハードウェア仕様

システム構成，本体および周辺装置のハードウェアについて，詳細に解説しました．周辺装置には，ディスク，プリンタなどの他にパソピアの特徴である，ROM/RAMPAC，液晶ディスプレイ等も含めました．

第2章　T-BASICの内部構造

T-BASIC(Ver 1.1)について，メモリ内部の状態，数値表現のされ方等を詳しく解説しました，また，ROM内ルーチンの使用例，ポインタの利用法も載せ，巻末の付録の利用の手引きとしました．

第3章　グラフィック

160×200フルカラー・グラフィック，高速GET@，PUT@，タイリング，WINDOW・VIEW

機能の追加など，T-BASIC(Ver1.1)のグラフィック機能を引き出すノウハウを載せました．

第4章　入出力装置

T-BASICを対象に，カセット，ディスク等のフォーマットや，ファイルの入出力について解説し，使用の際知っておくと便利なルーチンを載せました．特に，パソピアの特長である，RAMPACについては，別に項を立て解説してあります．

第5章　キー入力

T-BASIC(Ver1.1)を対象に，プログラマブル・ファンクションキー，キー割込，リアルタイム・キースキャン等キー入力についての解説し，合わせて各入力命令の比較をしました．

第6章　漢字出力

T-DISKBASIC(Ver1.0)に含まれた漢字ファイルや，漢字ROMPAC-2を使って，その内部表現や，出力の方法を解説しました．

第7章　OA-BASICの内部構造

東芝が開発した，OA-BASIC　Ver.D1.0(ディスク版)の特徴を，内部構造まで含めて解説しました．また，他に比べると強化されたファイル機能については，特に項を立て説明しました．

第8章　MINI-PASCAL

ROMPACで供給される，パソピア・MINI-PASCALについて，概要とその構造，プログラムの入力，実行例を載せました．また，内部ルーチンの一覧表を巻末のAPPENDIXに含めました．

第9章　CP/M

パソピアのCP/Mの概要と，特徴について解説し，いくつかのコマンドの使用例を載せました．いまや，8ビットCPUマシンの代表的DOS(ディスク・オペレーティング・システム)となったCP/Mを，初歩から知りたい方の参考になると思います．

目次

第1章　ハードウェア仕様

第2章　T-BASICの内部構造

第3章　グラフィック

第4章　入出力装置

第5章　キー入力

第9章 CP/M

APPENDIX

第1章

ハードウェア仕様

第1章　ハードウェア仕様

1-1　システム構造

1-1-1　概要

パソピアは，最新の電子技術，コンピュータ技術を駆使して作られ，比較的低価格でありながら，640×200ドットの高解像度グラフィック，サウンド機能など，数々の特徴を持ったコストパフォーマンスの高いパーソナルコンピュータです．

パソピアのシステムは，すべて直線を基調とし，色もダークシルバーに統一され，オフィスコンピュータを意識したデザインがうかがえます．

写真1-1
パソピア本体

パソピア本体は，いわゆる〝くさび形〟をしており，本体のケースは，裏側にある1本のネジによって固定されています．そのネジを取り除けば，あとは本体上部と下部が，はめ殺しになっているだけなので，比較的容易に分解することができます．周辺機器も，比較的シンプルな部品構成になっており，生産性の向上が計られています．これは，従来のパーソナルコンピュータより，製品として完成度が向上したといえるでしょう．システム構成図を，図1-1-1に示します．

本体のCPUには，4MHzのZ80Aが使用され，I/O用にZ80ファミリーのPIO(パラレル・インプット・アウトプット・インタフェィス)と3個の8255を用いDMAを全く行なっていないため，スループットの向上が計られています(1-2パソピア本体の仕様参照)．

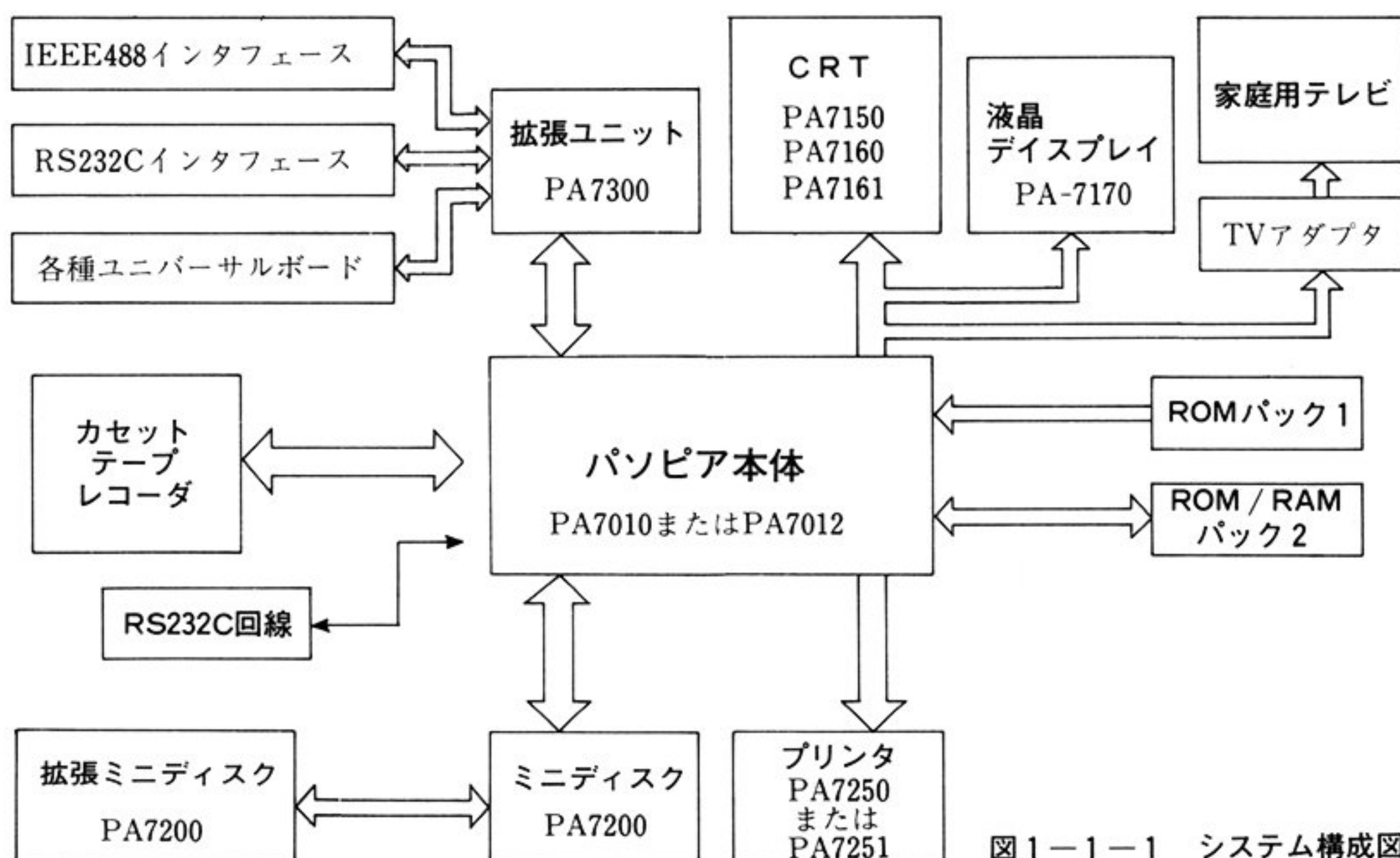



図1－1－1　システム構成図

メモリは，ROM32Kバイト，RAM64Kバイトが標準実装され，16KバイトのV-RAMと合わせて112Kバイトもの容量を持っています．

また，オプションのカートリッジを装着するだけで，ROM，RAM合計で64Kバイトまでを簡単に拡張することができる設計になっており，これらのメモリは，システムに応じてI/Oポート経由でバンク切換等により使用されます．（1－2，1－7参照）

1－1－2　拡張性

システムの拡張性についてはよく考えられており，ディスク，プリンタ等はもちろん，RS-232C，液晶ディスプレイ装置，ROM/RAMカートリッジ等も外部インタフェィスなしで，本体に接続することができます(写真参照)．

写真1－2
システム全体

特に，ROMカートリッジをつけることによって，メインメモリの下位(0000～7FFFH)32Kバ

イトがカートリッジ上のROMに置き変わるため，本体内にあるBASIC以外の言語(例えばパスカル等)を，待ち時間なしで使用することができます.

ROMカートリッジを使用しなくても，メモリのバンク切換によって，64KバイトRAMになるので，ディスクシステムなら，あらゆる言語を扱うことができる設計になっています(OSとしてCP/Mが用意されています).

基本ソフトウェアと，それを使用するためのシステムの関係を図1-1-2に示します.

基本ソフトウェア	本体の他に必要な最少限のシステム	著作権元
T-BASIC ver 1.0, 1.1	○CRTディスプレイ(家庭用テレビも含む) または液晶ディスプレイ ○カセットレコーダ	マイクロソフト
T-DISK BASIC ver 1.0, 1.1	○ディスプレイ ○ミニディスク・ユニット	マイクロソフト
OA-BASIC	○ディスプレイ ○カセットレコーダ	東芝
OA-DISK BASIC	○ディスプレイ ○ミニディスク・ユニット	東芝
MINI-PASCAL	○カセットレコーダ ○ROMパック ○ディスプレイ	東芝
UCSD-PASCAL	○ディスプレイ ○ディスク・ユニット	カリフォルニア大学理事会
CP/M	○ディスプレイ ○ミニディスク・ユニット	デジタルリサーチ

図1−1−2　基本ソフトウェアとシステム

1-1-3　システムの起動

電源が投入された際，周辺機器の有無を自動的にチェックし，それに応じてシステムが起動します.

ROMカートリッジが装着されている場合は，ROMカートリッジの方が優先されます．また，液晶ディスプレイ装置がついている場合は，テキスト画面が液晶ディスプレイの方に出力されます.

それでは，それぞれのシステムの起動について説明しましょう.

・T-ROMBASICの場合.

メインメモリの下位32Kバイト(0000～7FFFH)がROMに割りあてられ，上位32KバイトがRAMになります．この時，下位32KバイトのRAMは使用されておりません．メモリの構成図を，図1-1-3に示します.

また，OA-BASIC仕様のパソピアにT-BASICのROMカートリッジを装着した場合も，全く同様に作動します．

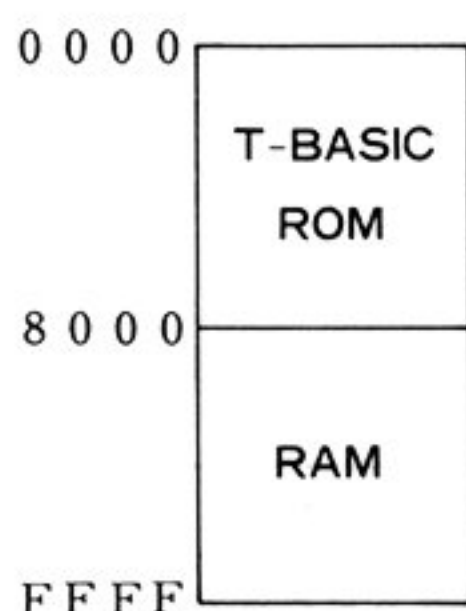



図1—1—3　T-BASIC起動時

・T-DISKBASICの場合

電源投入時，フロッピィディスクユニットが接続されていると，まずシステムディスクの入っているドライブを探します．ディスクドライブに，ディスケットが入っていなかったり，T-DISKBASICのシステムディスクがない場合は，ROMBASICが立ち上がり，ディスクドライブは無視されます．ドライブ-1または2のいずれかに，システムディスクが入っていると，まず，ROMの内容の一部がRAMに転送された後，バンク切換が行われ，次いで，DISKBASICがシステムディスクより書き込まれます．

T-DISKBASICでは，64Kバイトのメインメモリがすべて RAMとなり，T-BASICのROMは全く使用されなくなります．このときのメモリ構成を図1-1-4に示します．

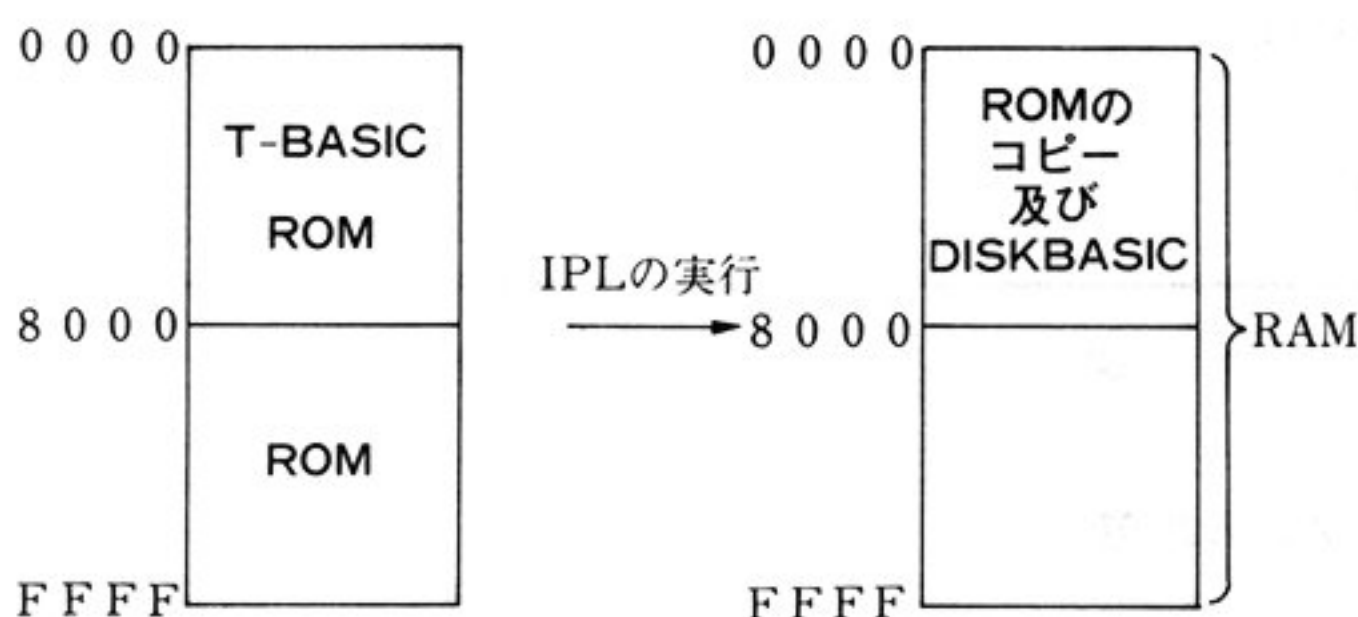



図1—1—4　T-DISKBASIC起動時

・OA-ROMBASICの場合

OA-ROMBASICが起動されると，まずBLT(Basic Logic Test)という，システムの簡易テストが行われます．そしてROMの0000H番地から7200H番地までが，同じアドレス上にあるRAMに転送され，ついで，バンク切換が行われ，64KRAMモードでOA-BASICが起動されます．しかし，RENUM，CLOAD，CSAVE，LOAD，SAVE，TERMの各コマンドの実行は，バンクがROMに切り換わり，ROM上で実行されます．これは，これらの各コマンドの処理ルーチンの入っている7200H番地から7FBFH番地までを，RAM上でデータエリアとして使用しているためです．図1-1-5にメモリ構成図を示します．

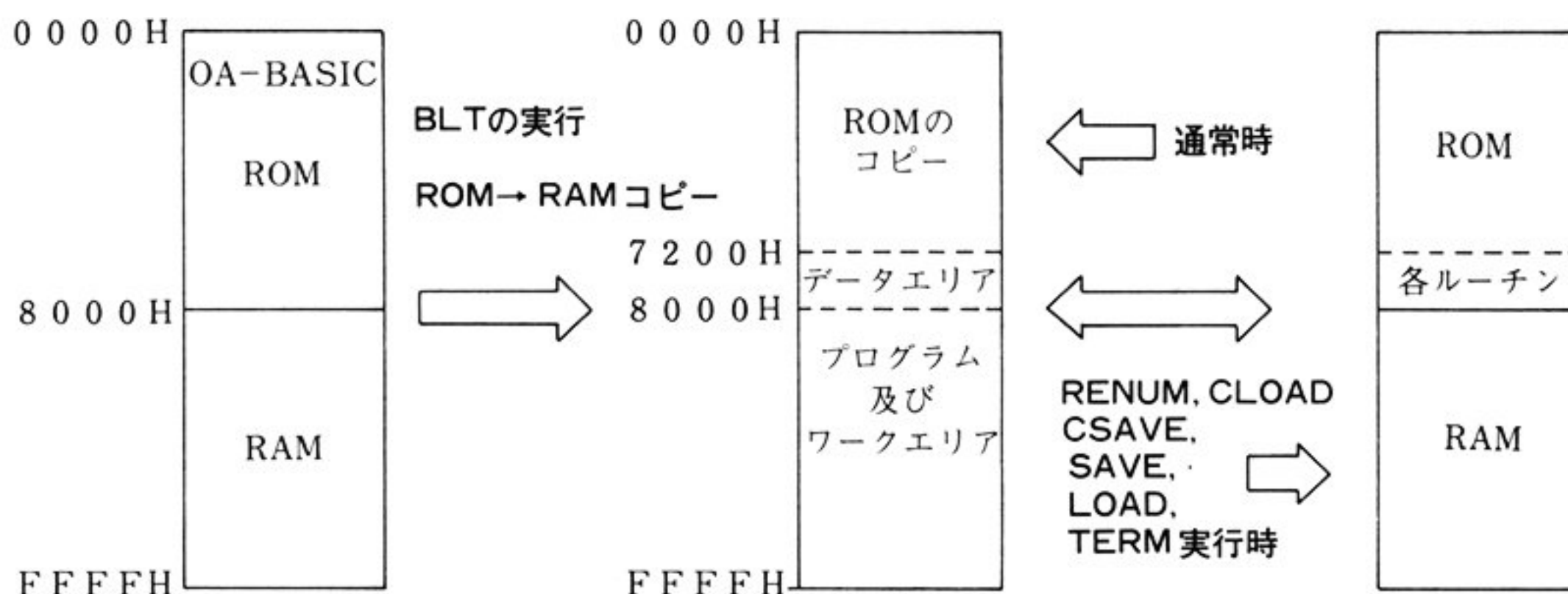



図1−1−5　OA-ROMBASIC起動時

・OA-DISKBASICの場合

OA-DISKBASICが起動すると，まずBLTを実行し，RAM上にDISKBASIC本体がディスクより転送されます．そして，ディスク関係のコマンドはRAM上で，その他のコマンドはROM上と，そのつどバンク切換を行って実行します．図1-1-6にメモリ構成図を示します．

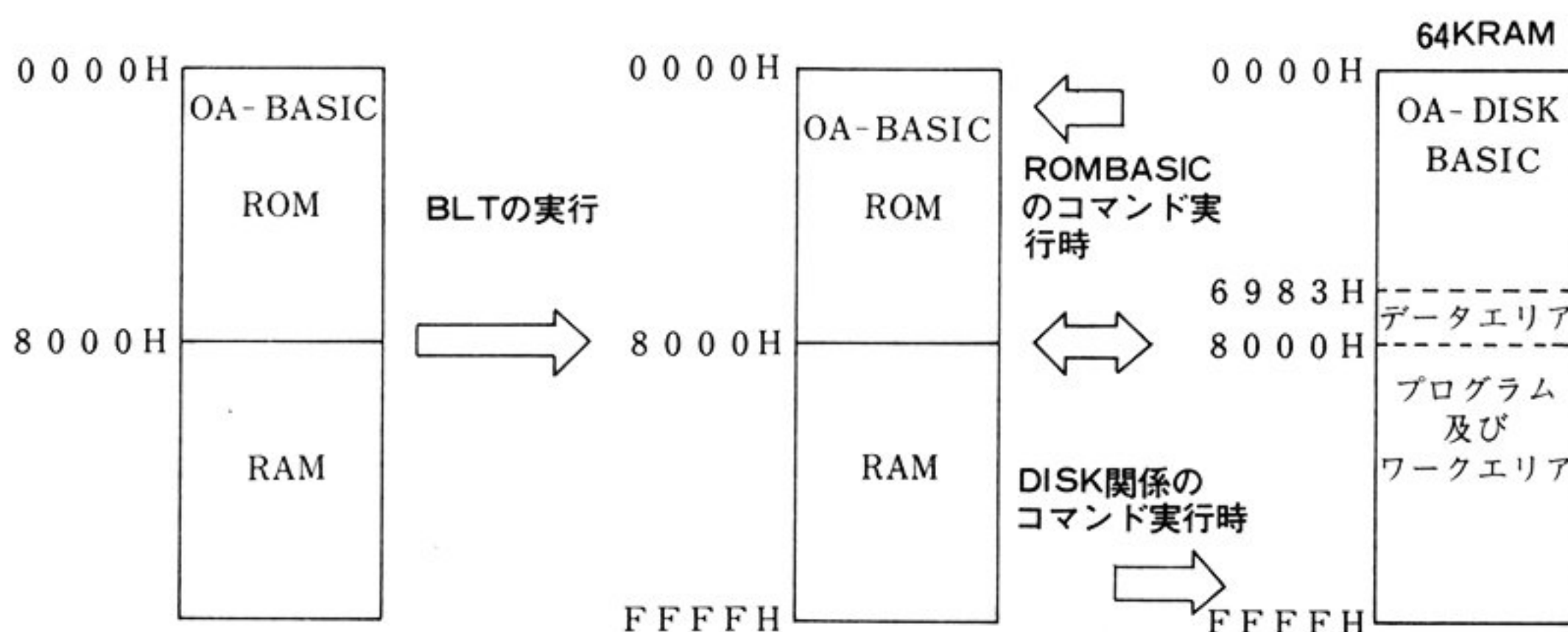



図1−1−6　OA-DISKBASIC起動時

その他，パスカル，CP/M等については，それぞれ第8章，第9章を参照して下さい．

システム起動後，リセットスイッチが押されると，そのときのメモリバンクの番地から再起動します．もし，ROMモードならば，電源投入時と同じ動作が行われます．

1-2　パソピア本体の仕様

1-2-1　本体ブロック図

本体ブロック図を図1-2-1に示します．

15.9744
OSC
4 MHz
CPU
Z 80 A
RAM
64KB
4164
デコーダ
マルタ
プレクサ
157
ROM
32KB
2364
ROMパック 1
ROM/RAMパック 2
バッファ
245
バッファ
367
データ　バス
バッファ
245
外部
バス
バッファ
367
アドレスバス
デコーダ
バッファ
373
PPI
8255
PIO
CTC
CRTC
465055
PPI
8255
PPI
8255
VRAM
16KW
4116
キャラクタ
ジェネレータ
2716
プリンタ
インタ
フェース
RS232C
インタ
フェース
オーディオ
カセット
インタ
フェース
キーボード
スピーカ
CRT
インタ
フェース

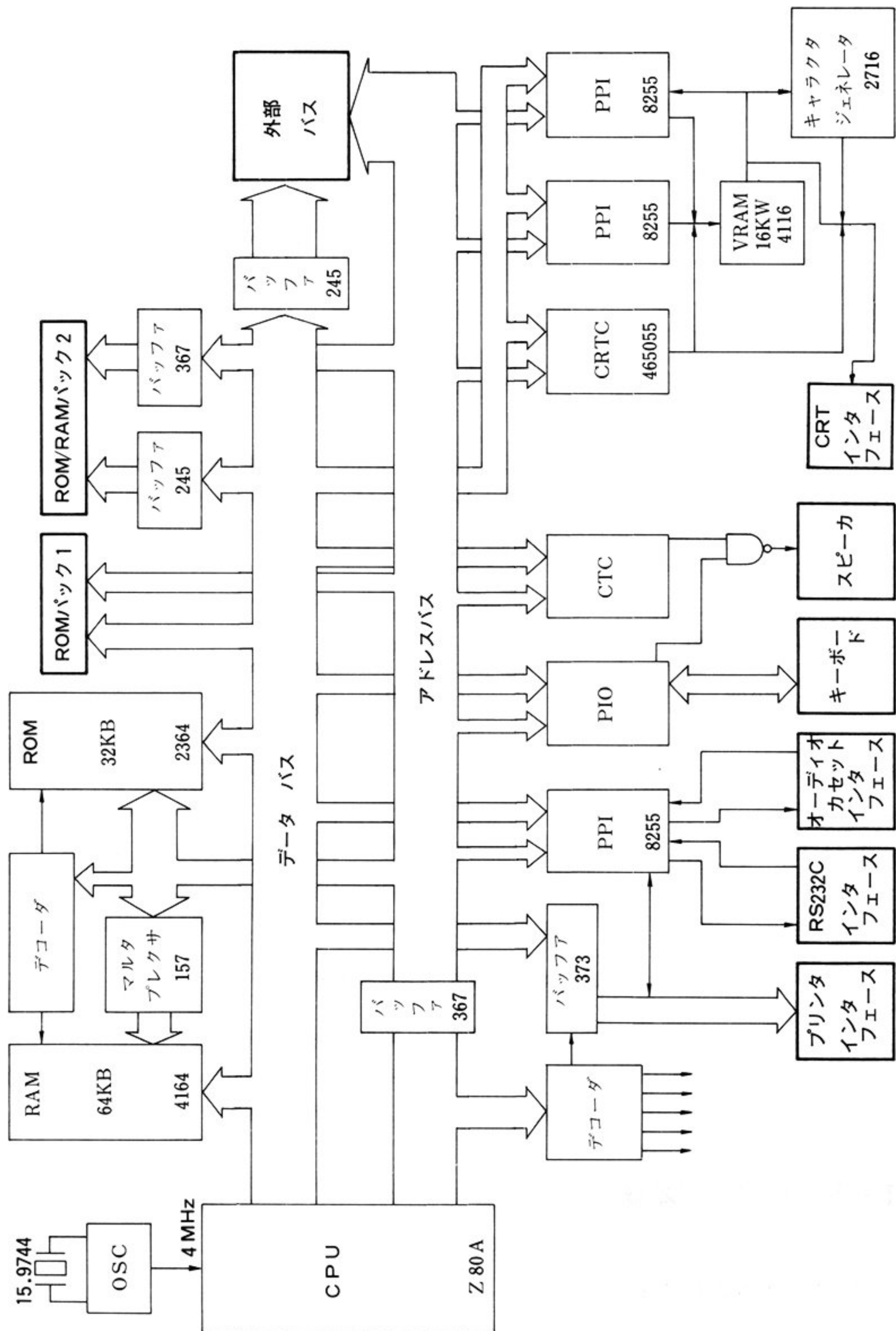



図1―2　本体ブロック図

1-2-2　チップ構成

CPUにはZ80A(4MH$_z$)を使用しており,画面処理からI/O関係まですべての事を行っています.メインメモリは,RAM64Kバイト,ROM32Kバイト標準装備で,ROMを32Kバイト拡張でき,このうち,RAM64Kバイト,ROM64Kバイトを,バンク切換によってメインメモリ上におくことができます.

他に本体のスロットにメモリを差し込むだけでZ80のI/O空間に自由にメモリを置くことができます.現在のところ96KROM,16KRAMが市販されていますが,さらに拡張が可能になっています.このメモリの入出力はI/O経由で行なわれます.(1-7参照)

V-RAM(ビデオRAM)は9bit×16kという構成で,メインメモリ上ではなく,I/Oポート経由で読み書きが行われています.そのため,画面を表示するのにDMAを用いる必要がなく,CPUの持ち時間が少なくなっており,CPUのスループットが向上しています.1ワードが9ビットなのは,データが8ビットで,キャラクタ,グラフィック判別用のアトリビュートを1ビット持っているためです.

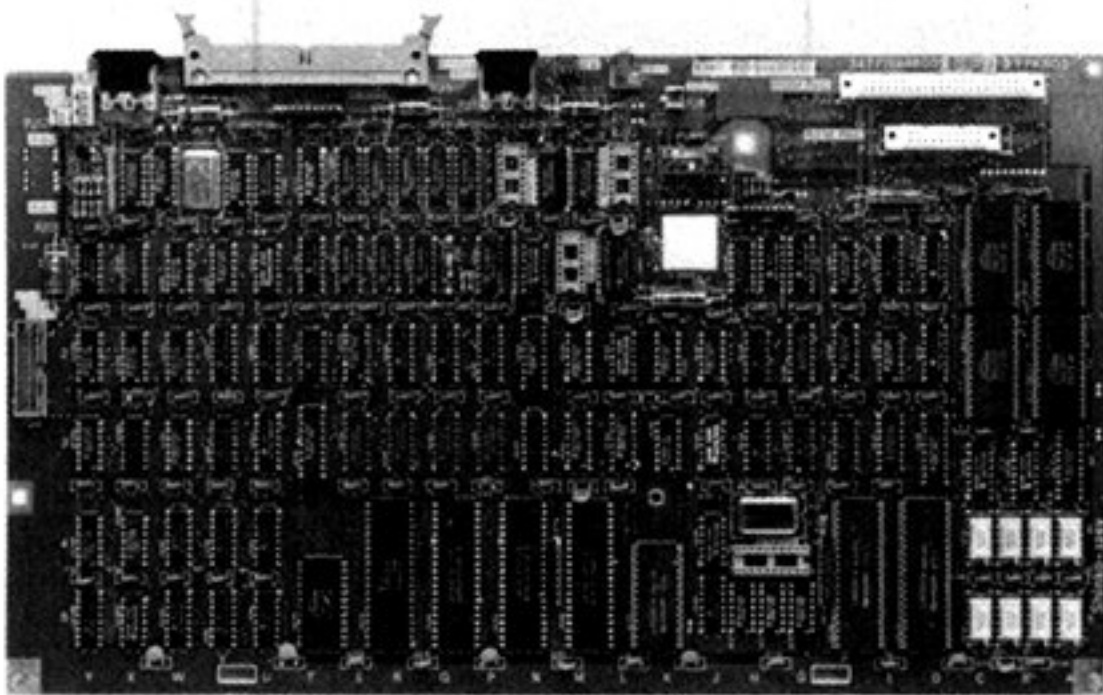



写真1-3
チップ構成

メモリの構成図を,次のページ図1-2-1に示します.

その他に,Z80AファミリーのPIO,CTCや,PPI(8255),そしてCRTC(HD465055)等によって構成されており,割込式キーボードコントローラー,4個の独立したタイマ,ダイナミックメモリのリフレッシュ,各種I/Oの制御を実現しています.(写真1-4)

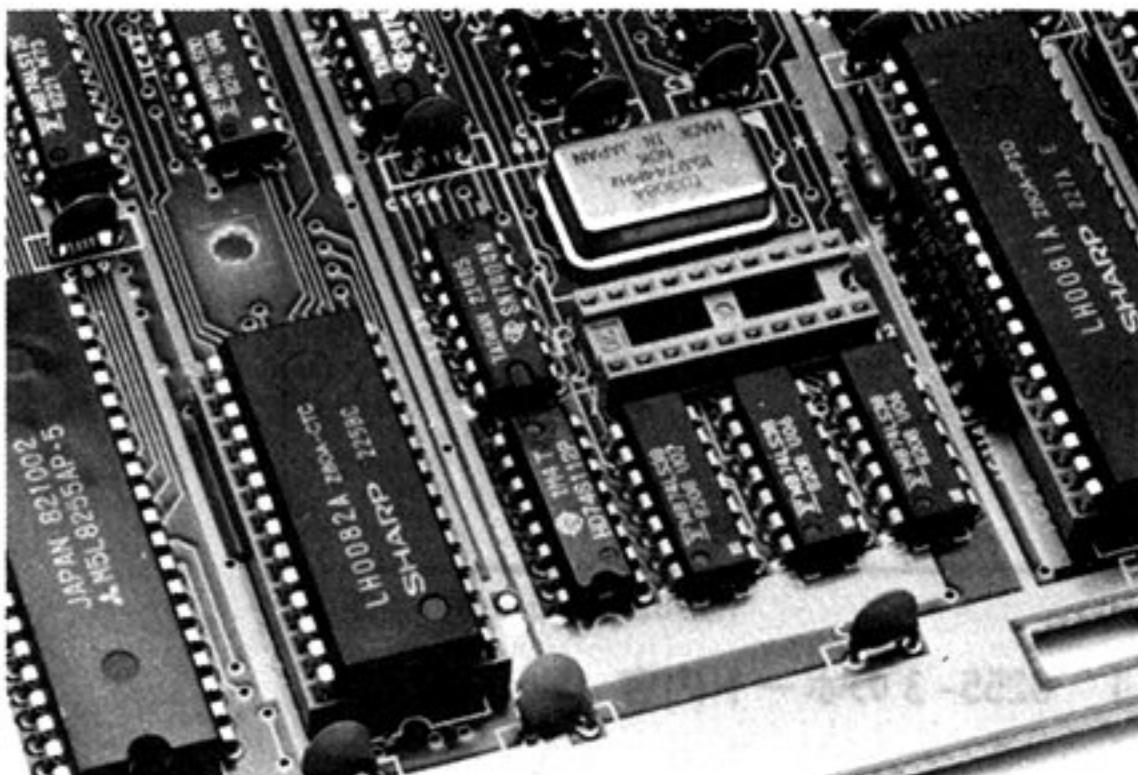



写真1-4
Z80ACTC(カウンタ・タイマ)

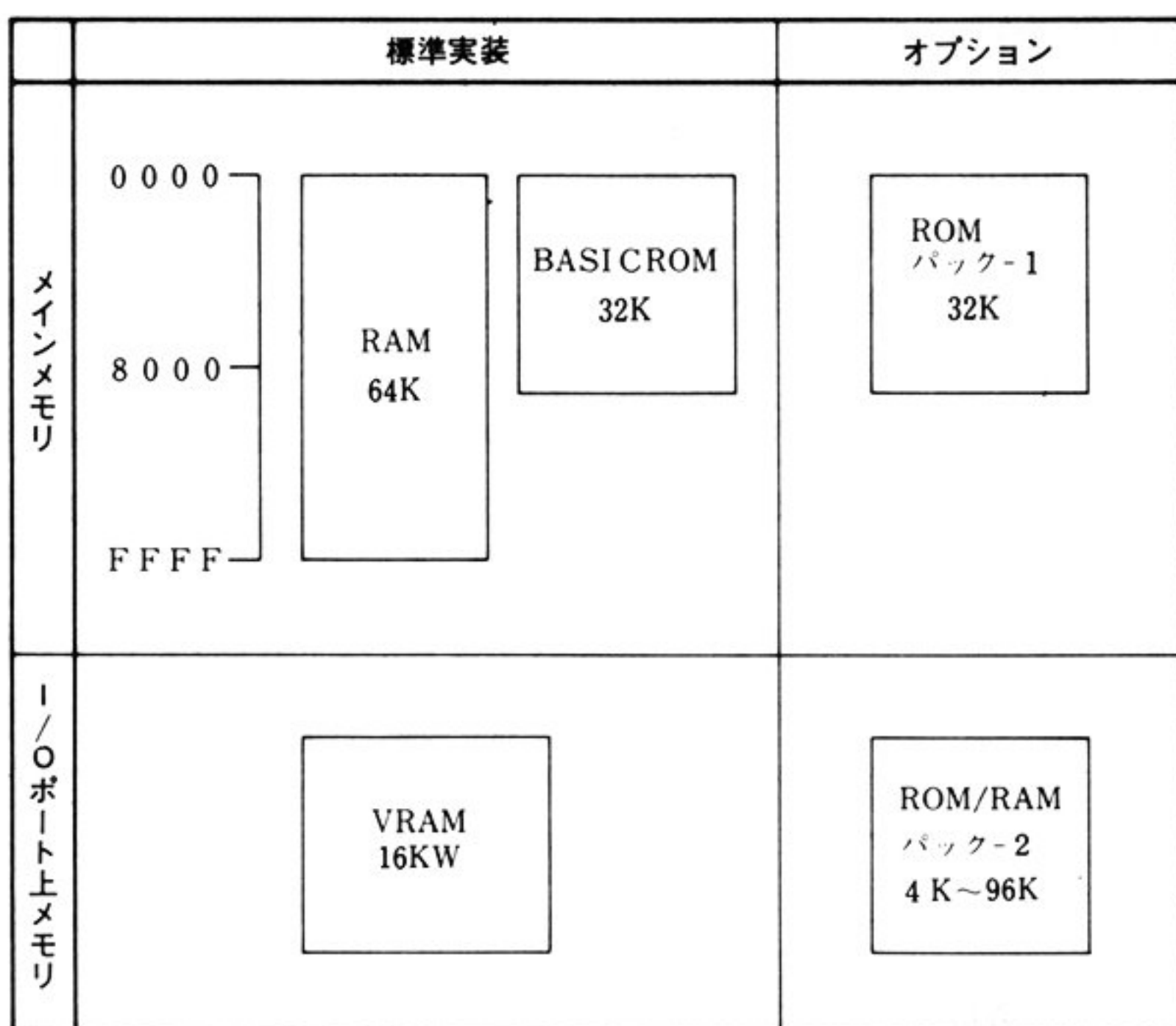



図1―2―1　メモリ構成図

ポート	動作モード／ポート番号	端子	アクテイブ	コントロール内容
A	出力20H	PA7	H	プリンタへPRIME信号を出力します
	〃	PA6	H	〃　STROBE　〃
	〃	PA5	L	カセットのリモート端子用リレーのON/OFF
	〃	PA4	−	カセットデータの出力
		PA3		
		PA2		
		PA1		
		PA0		
B	入力21H	PB7	L	プリンタのFAULT信号入力
	〃	PB6	H	〃　BUSY　〃
	〃	PB5	−	カセットデータの読み出し
	〃	PB4	L	RS232CのCIまたはTXC信号の入力
	〃	PB3	L	〃　DCD信号の入力
	〃	PB2	L	〃　DSRまたはRXC信号の入力
	〃	PB1	L	〃　CTS信号の入力
	〃	PB0	H	〃　RXD信号の入力
C	出力22H	PC7	L	RS232CのSTI信号の出力
	〃	PC6	L	〃　DTR　〃
	〃	PC5	L	〃　RTS　〃
	〃	PC4	H	〃　TXD　〃
	入力22H	PC3	H	ROMPACモード入力
	〃	PC2	H	RAMモード入力
	〃	PC1		
	〃	PC0		

〈注〉8255はモード0で使用

図1―2―2―1　8255-3のポート内容

Z80A-PIO(パラレルインプット/アウトプット)は，キーボードスキャン用信号の出力，キーボードデータの入力，およびスピーカのON/OFF信号の出力を行っています．（１-２-４参照）

PPI(プログラマブル・ペリフェラル・インタフェイス)である8255は３個使用され，そのうち２個をV-RAM，CRTまわりの制御に，残りの１個をカセット・インタフェイス，プリンタ，RS-232C，バンク切換に用いています．図１-２-２-１，２にその接続図とポート内容を示します．（１-３，１-６，１-９参照）

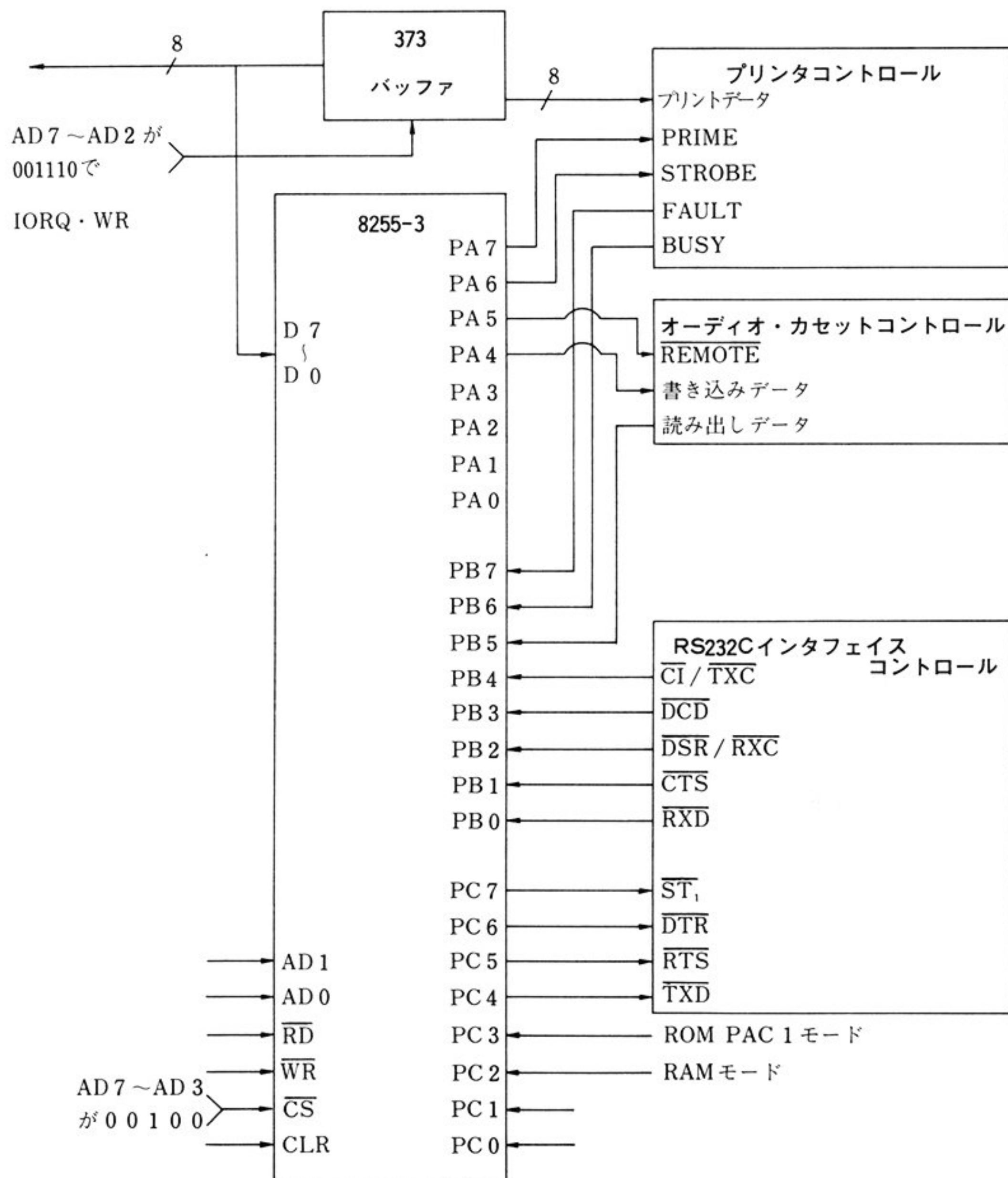
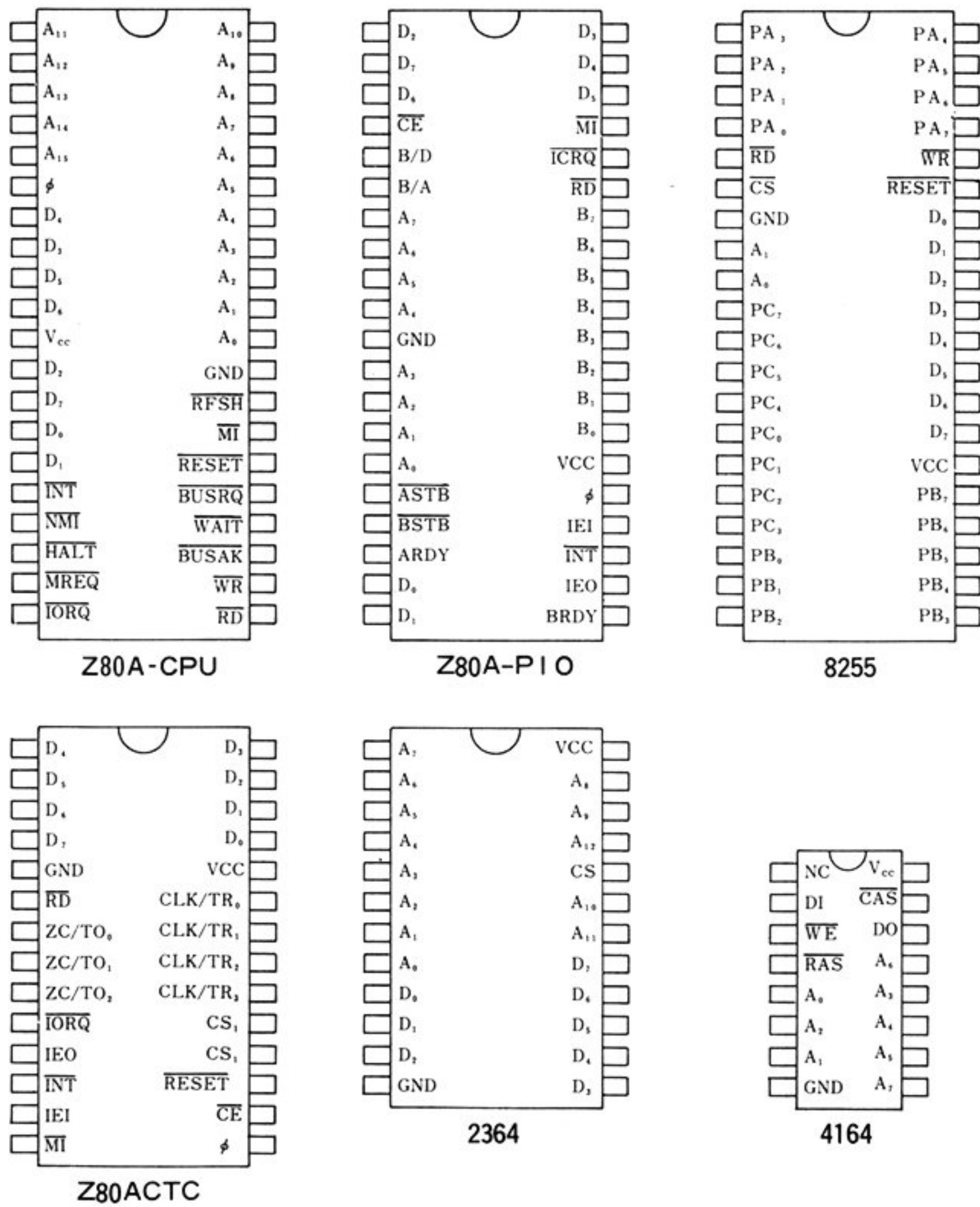
図１－２－２－２　8255-3まわりの接続図

次に，パソピア本体で使用されている，チップのピン配列を示します.

図１－２－２－３　各種チップピン配列

1-2-3　バンク切換

メインメモリのバンク切換は，I/Oポートの3CHにデータを出力することによって行います．図1-2-3に，このI/Oポートのデータ内容を示します．

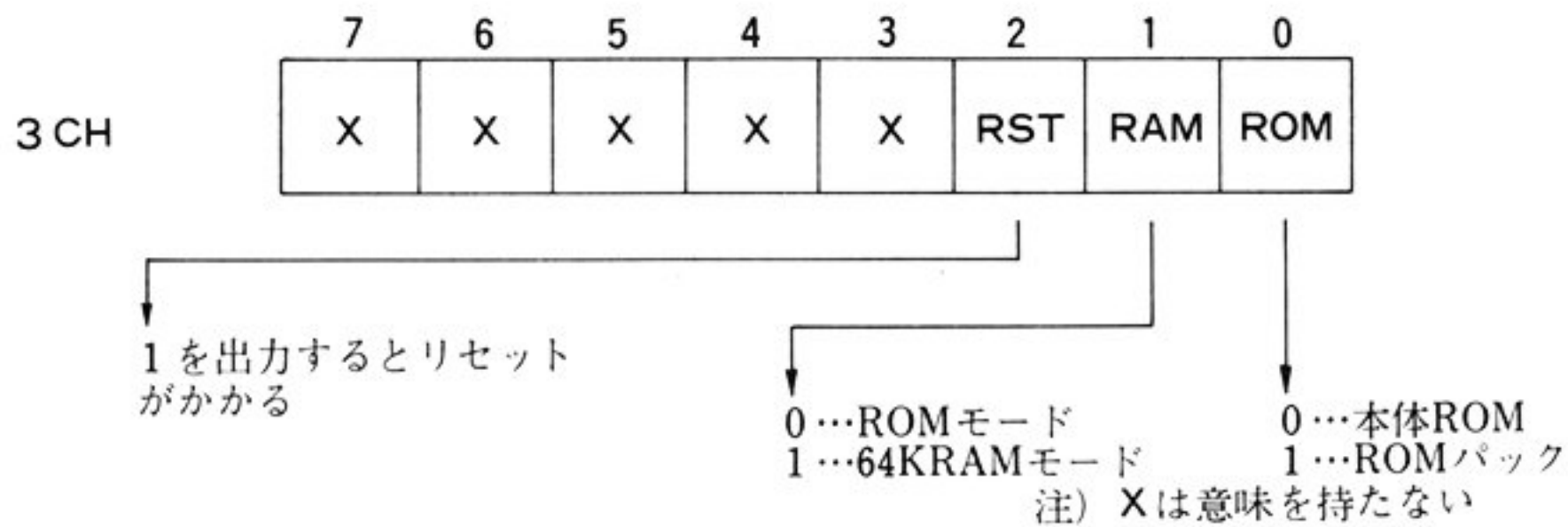



図1－2－3　バンク切換

このようにデータをセットして，OUT命令によってバンク切換を行うとき，出力データのビット2が0であれば，新しいバンク上の次のアドレスから実行されますが，ビット2が1の場合，ハード的にリセットがかかり，新しいメモリバンク上の0番地から実行されます．

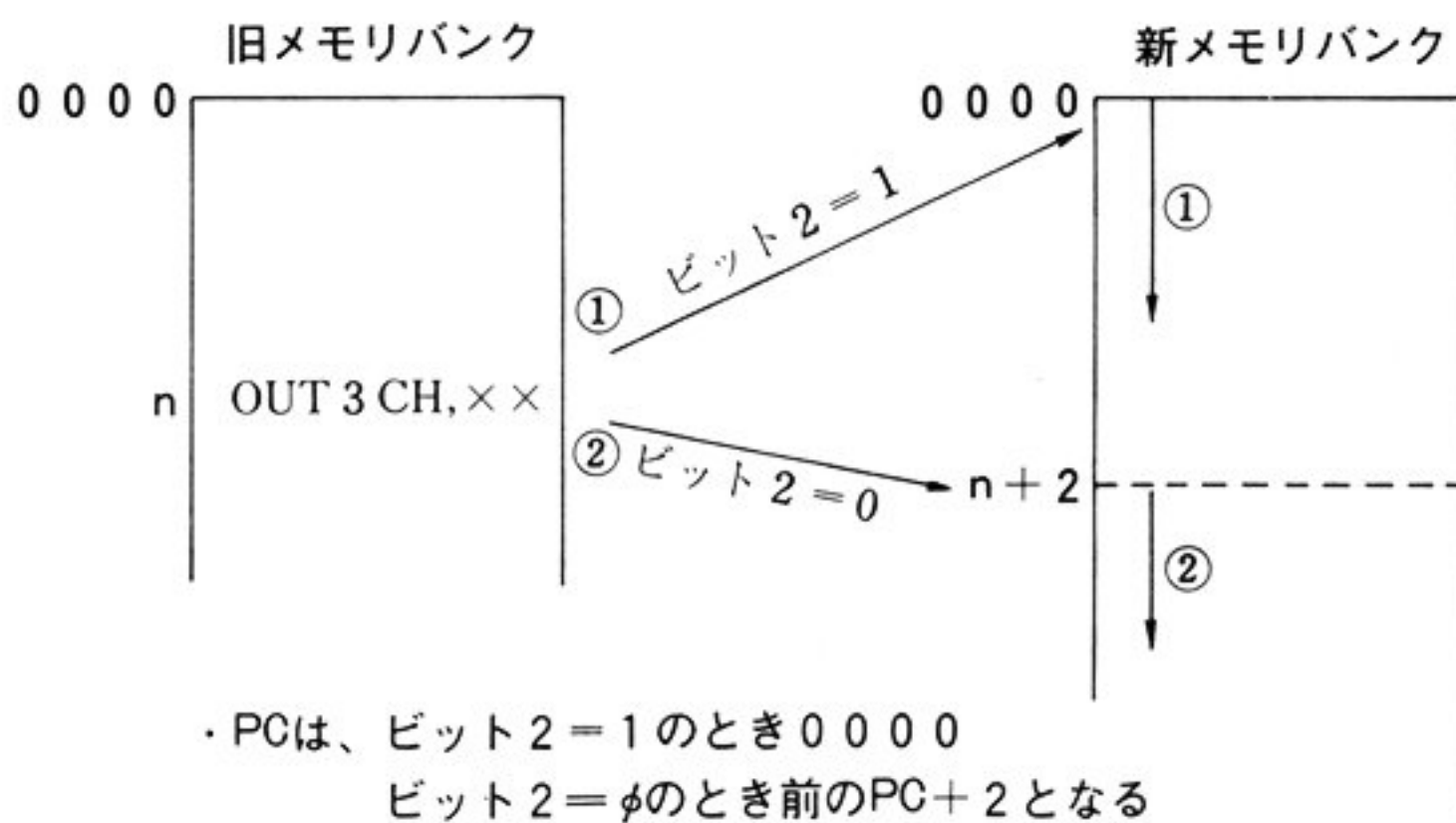



図1－2－4　PCの変化

現在のメモリモードは，I/Oポートの22H（ポート）を読むことによって認識できます．図1-2-5に，このポートのデータ内容を示します．

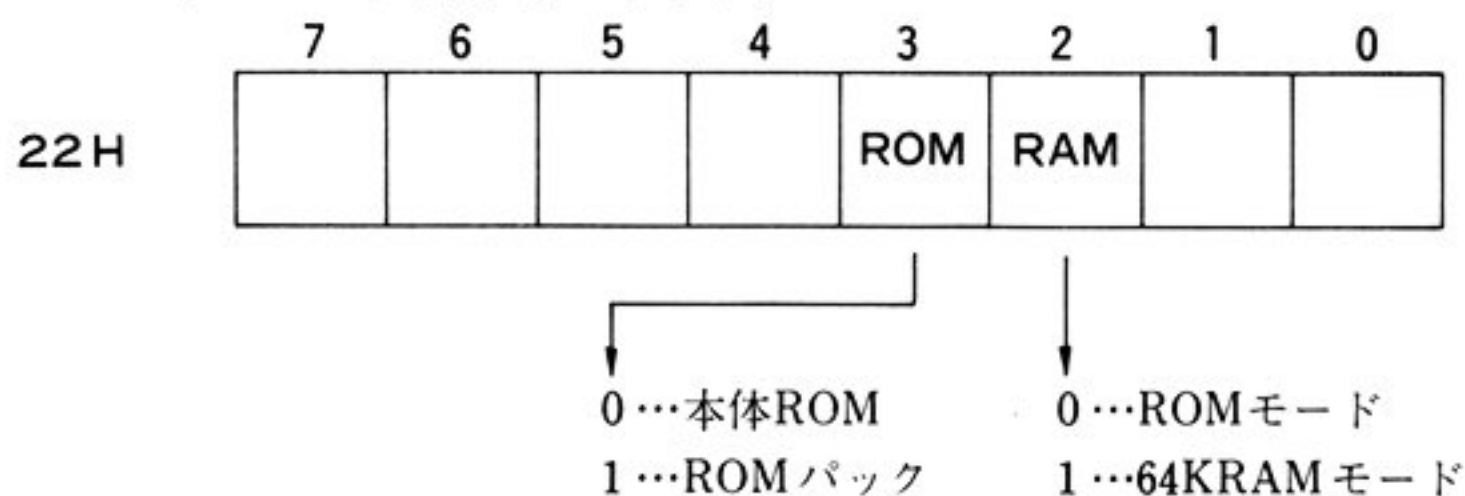



図1－2－5　メモリバンクの読み出し

3種類のメモリバンクの各状態を図1-2-6に示します.

ROMPAC-1が選択された場合, ROMの実装容量によってRAMの容量が32K～56Kバイトになります. また, メモリバンクがROMになっている場合に, データを書き込むと, 同一アドレスのRAM上に書き込まれるようになっています. つまり, RAMへの書き込みは, バンクに関係なく行うことができます. ただし, そのままでは読み出しはできません. (2-1-9参照)

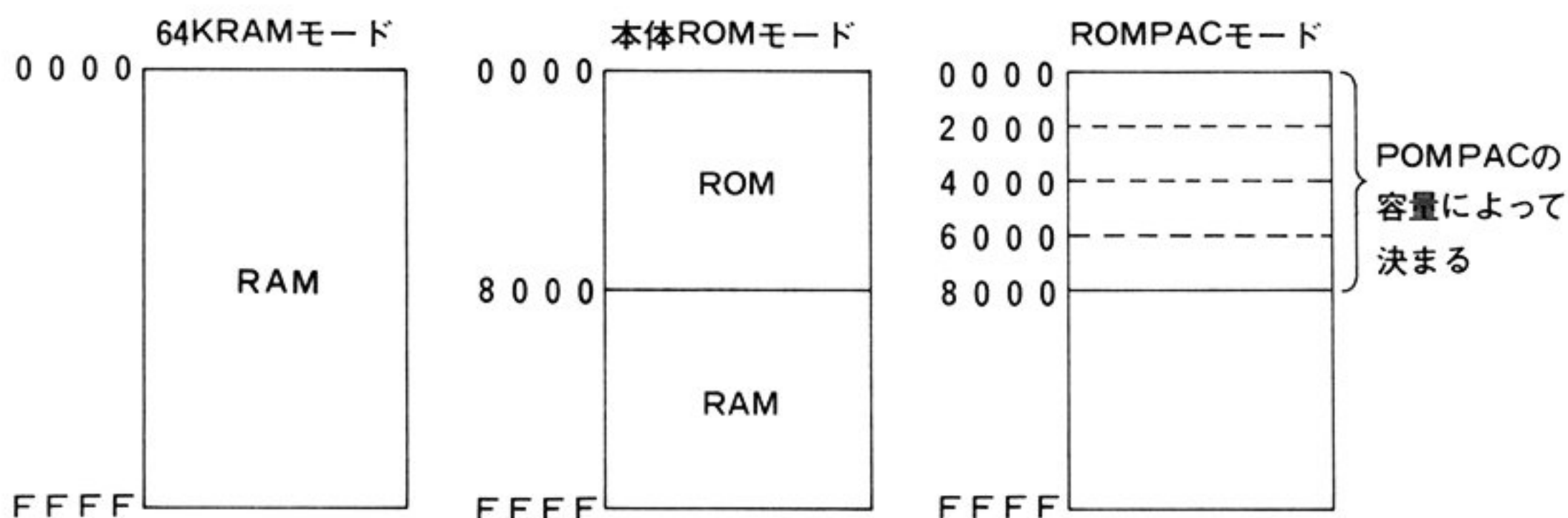



図1—2—6　3種類のメモリバンクモード

1-2-4　キーボード

キーボードの制御は, Z80AファミリのZ80APIOというパラレルインタフェイス用のチップを用いる事により, 効率的に行っています.

このチップは8ビットのI/OポートA, B, 2つ持っており, ポートBのいずれかの端子をLOWにすることにより, 割込信号を発生させています.

この機能を生かしてキーボードは割込式となっており, いずれかのキーが押されたとき, 割込信号が発生し, CPUにキーが押されたことを知らせます. そのため, キーが押されていないときは, キーボードスキャンが行われず, その結果CPUの負担を減らしています.

キーボードスキャンは, ソフトウェアスキャン方式で, PIOのポートAにスキャンラインを出し, ポートBでデータを読んでいます. また, ポートAのビット7は, スピーカーのON, OFFスイッチとなっています. またこのキーボードは, CAPSLOCKやカナキーが, 機械式ロックになっておらず, その切り換えはすべてソフトウェアで行っています.

図1-2-7に, PIOキーボードの接続図を, 図1-2-8, 9にキーボードマトリクスとキーボード用ソケットの図を示します.

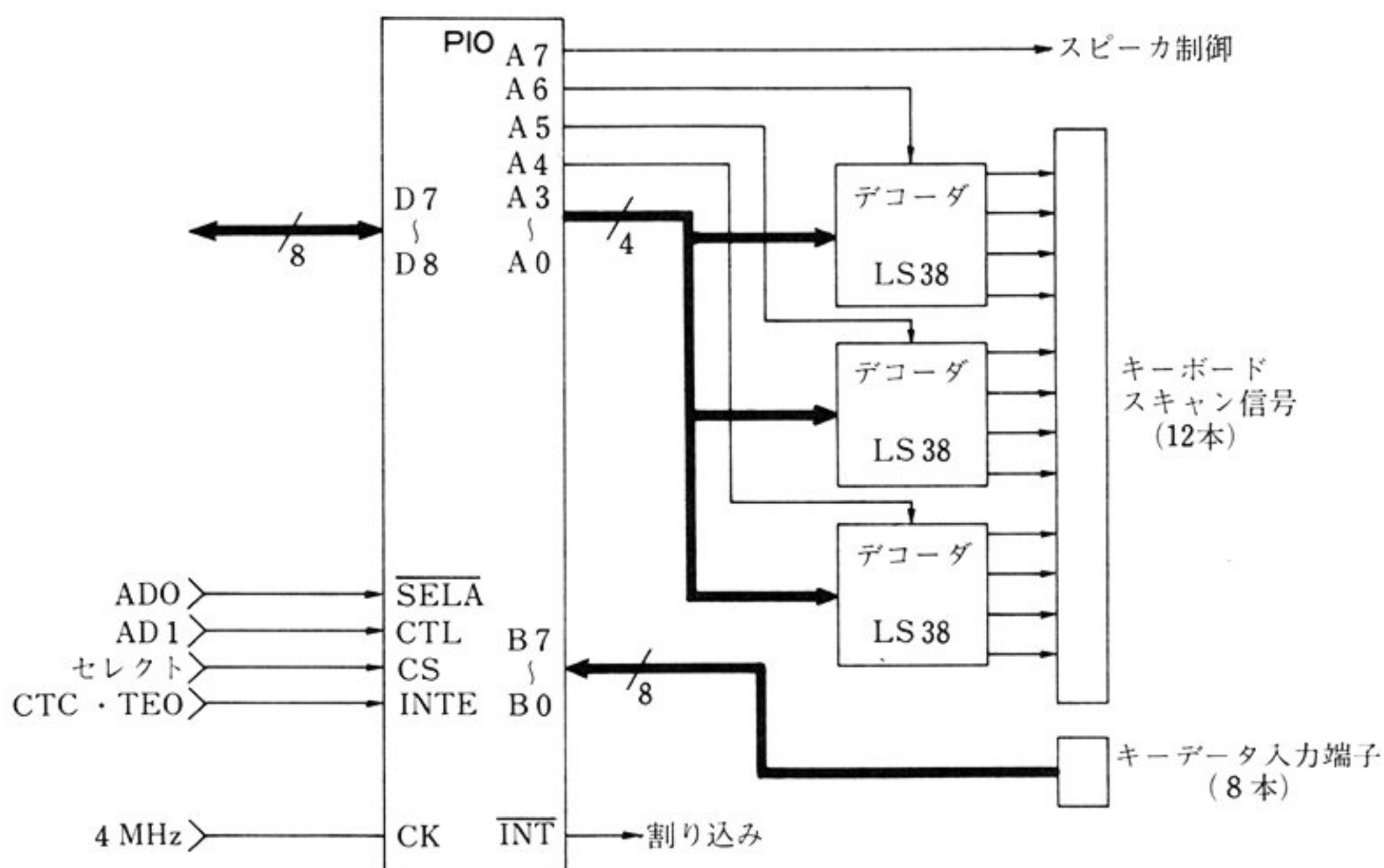



図1－2－7－1　PIOとキーボードの接続図

ポート	動作モード	端子	アクティブ	コントロール内容
A	出力モード3 (ビットコントロール) 割り込み無 ポート番号：30	A7	H	スピーカ発振を有効とします
		A6	H	スキャンブロックCを有効とします
		A5	H	〃　B　〃
		A4	H	〃　A　〃
		A3	H	名スキャンブロック内のそれぞれのスキャンラインを設定します
		A2	H	
		A1	H	
		A0	H	
B	入力モード3 (ビットコントロール) LowレベルのOR 条件により割り込みを 発生する ポート番号：31	B7	L	スキャン結果のデータ入力
		B6	L	
		B5	L	
		B4	L	
		B3	L	
		B2	L	
		B1	L	
		B0	L	

図1－2－7－2　PIOのコントロール内容

図1－2－8　キーマトリックス

ピン番号	信 号 名	I／O	ピン番号	信 号 名	I／O
1	$\overline{KR0}$	I	2	$\overline{KR1}$	I
3	$\overline{KR2}$	I	4	$\overline{KR3}$	I
5	$\overline{KR4}$	I	6	$\overline{KR5}$	I
7	$\overline{KR6}$	I	8	$\overline{KR7}$	I
9	$\overline{KS0}$	O	10	$\overline{KS1}$	O
11	$\overline{KS2}$	O	12	$\overline{KS3}$	O
13	$\overline{KS4}$	O	14	$\overline{KS5}$	O
15	$\overline{KS6}$	O	16	$\overline{KS7}$	O
17	$\overline{RS6}$	O	18	$\overline{KS9}$	O
19	$\overline{KSA}$	O	20	$\overline{KSB}$	O

I：入力
O：出力

図1―2―9　キーボード用ソケット

1-2-5　割込機能

割込機能には，3つの動作モードを持つマスク可能なものと，マスク不可能なNMI(Non Maskable, Interrupt)がありますが，NMIの方は，本体では使用しておりません．マスク可能な割込はモード2を使用し，ベクタテーブル方式の割込を行っています．ベクタテーブルは，FEF0H～FF00Hに割りあてられています．図1-2-10に割込ベクタテーブルの表を示します．

発生場所	割り込みベクタ	テーブルアドレス	用　途
CTCチャンネル0	F0H	FEF0H	RS232C，オーディオカセットの タイミング制御
チャンネル1	F2H	FEF2H	スピーカーの発信周波数設定 (割り込みなし)
チャンネル2	F4H	FEF4H	キーボードのセルフリピートの タイミング制御
チャンネル3	F6H	FEF6H	システム・クロック
PIOポートA	F8H	FEF8H	キーボードスキャン用(割り込みなし)
ポートB	FAH	FEFAH	キーボード　〃
外部割り込み	FFH	FEFFH	拡振インターフェースからの割り込み

図表1―2―10　割り込みベクタ・テーブル

オーディオカセット・インタフェィスのタイミングやシステムタイマーなど一定の周期を必要

とする処理はZ80A-CTC(カウンタ・タイマ)によるタイマ割込で行っています．このCTCは４つの独立したタイマカウンタを持っておりそれぞれスピードタイマ，スピーカートーン，キータイマ，システムロックに使用されています．図１-２-11にCTCの各チャンネルの用途を，図１-２-12にCTCまわりの接続図を示します．

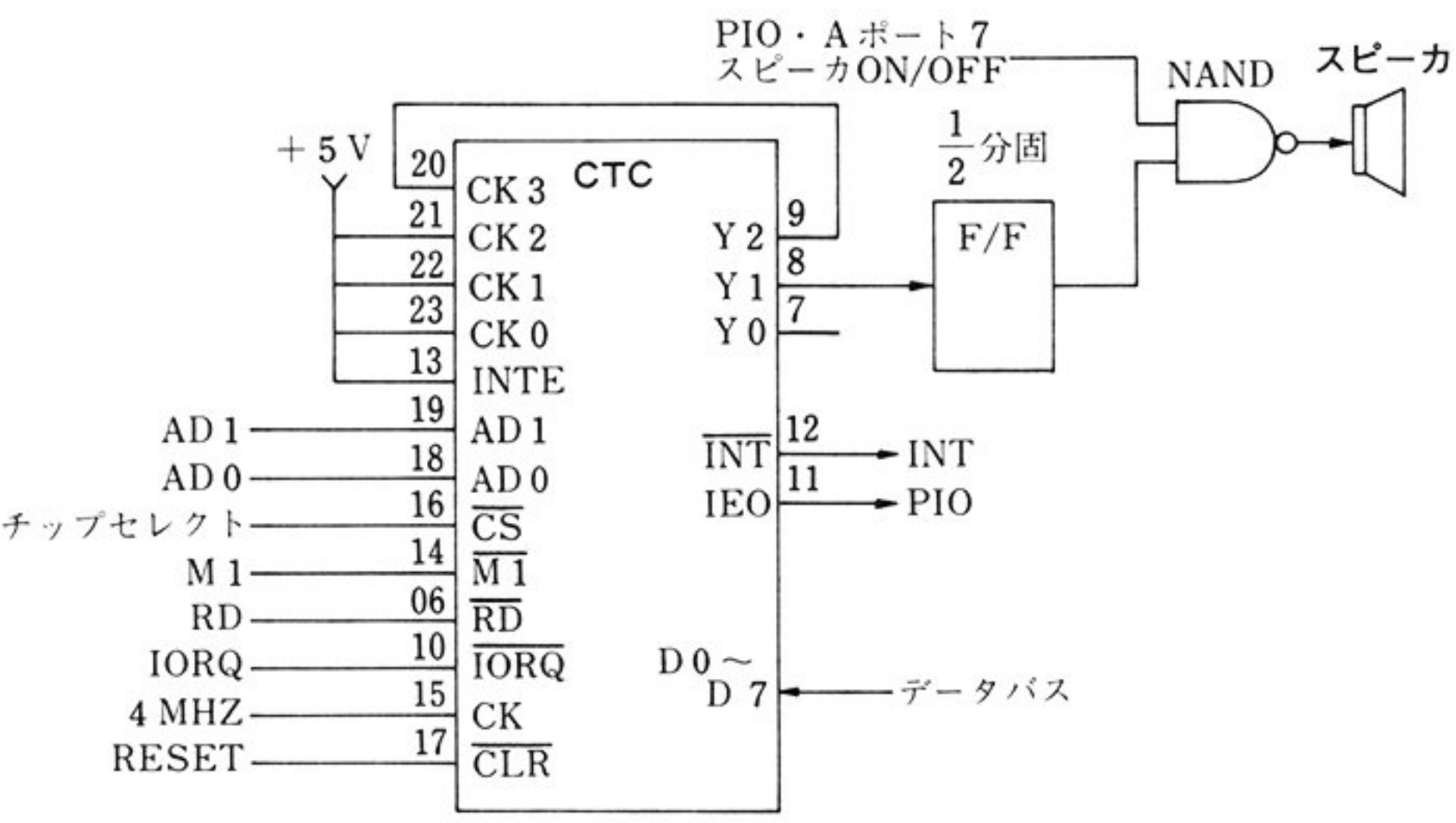



図１—２—12　CTCの接続図

チャンネル	モード	割り込み	用　　途
0	タイマ	有	スピード・タイマ オーディオカセット・インターフェースとRS232Cインターフェースで必要なスピード計時用のタイマ．
1	タイマ	無	スピーカ・トーン スピーカに供給する音程を決める周波数の発振器として用いる．キークリックやミュージックの音階に使われる．
2	タイマ	有	キーボード制御タイマ キーボード割り込み処理中のディバウンスとセルフーリピートの計時のために用いられる．したがってキーボードの処理はシステムクロックと独立に処理する．
3	タイマ	有	システム・クロック・タイマ システム・クロックを計時するためのタイマである．64Hzにセットされている．このため，スピーカにミュージックを出力するときの最小時間は，１／64分音譜となる．

図表１—２—11　CTC各チャンネルの用途

システムタイマは，約64Hzのタイマ割込によって制御されています．64Hzの割込周期はCTCのCH3にクロック(3.9936MHZ)を与え，それをプリスケーラとカウンタにより分周して得ています．これを式にすると，

$$3.9936\mathrm{MH_Z} \div \overset{(\text{プリスケーラ})}{256} \div \overset{(\text{カウンタ})}{244} = 63.934426\mathrm{Hz}$$

となります．パソピアではこの値を1/64秒としているため，１秒は，

$$\frac{1}{63.934426} \times 64 = 1.001256\text{秒}$$

になります．これでは１秒につき約１ミリ秒進むことになるので，正確な時計タイマを必要とする場合は，次の式によって得られる補正値をかけなければなりません．

$$\text{補正値} = \frac{1}{1.001256} = 0.9989754$$

その他の割込として，システムタイマを４分周した16Hzのソフトウェア割込があります．これはFECAH，FECBHに処理ルーチンの先頭アドレスを書き込むことによって使用します．

1-3　オーディオカセット・インタフェイス

1-3-1　概要

オーディオカセット・インタフェイスは，1600bps(bit-par-second)と比較的高速で，テープ走行速度の変動に強い，変調方式となっております．

この，オーディオカセット・インタフェイスの制御は，すべてソストウェアによって行われ，PPI(パラレル・インタフェイス)の8255-3から入出力を行っています．図1-3-1に，オーディオカセット・インタフェイスの接続図を示します．

論理０と１の書き込みは記録波形の周期によって決められ，それぞれ1200Hzと2400Hzによって表されます．この場合，転送速度が普通なら1200bpsになるのですが，1200Hzと2400Hzをそれぞれ１波形のみを出力するので，転送速度は，図1-3-2のような計算式により，1600bpsになります．このときの記録波形を図1-3-3に示します．

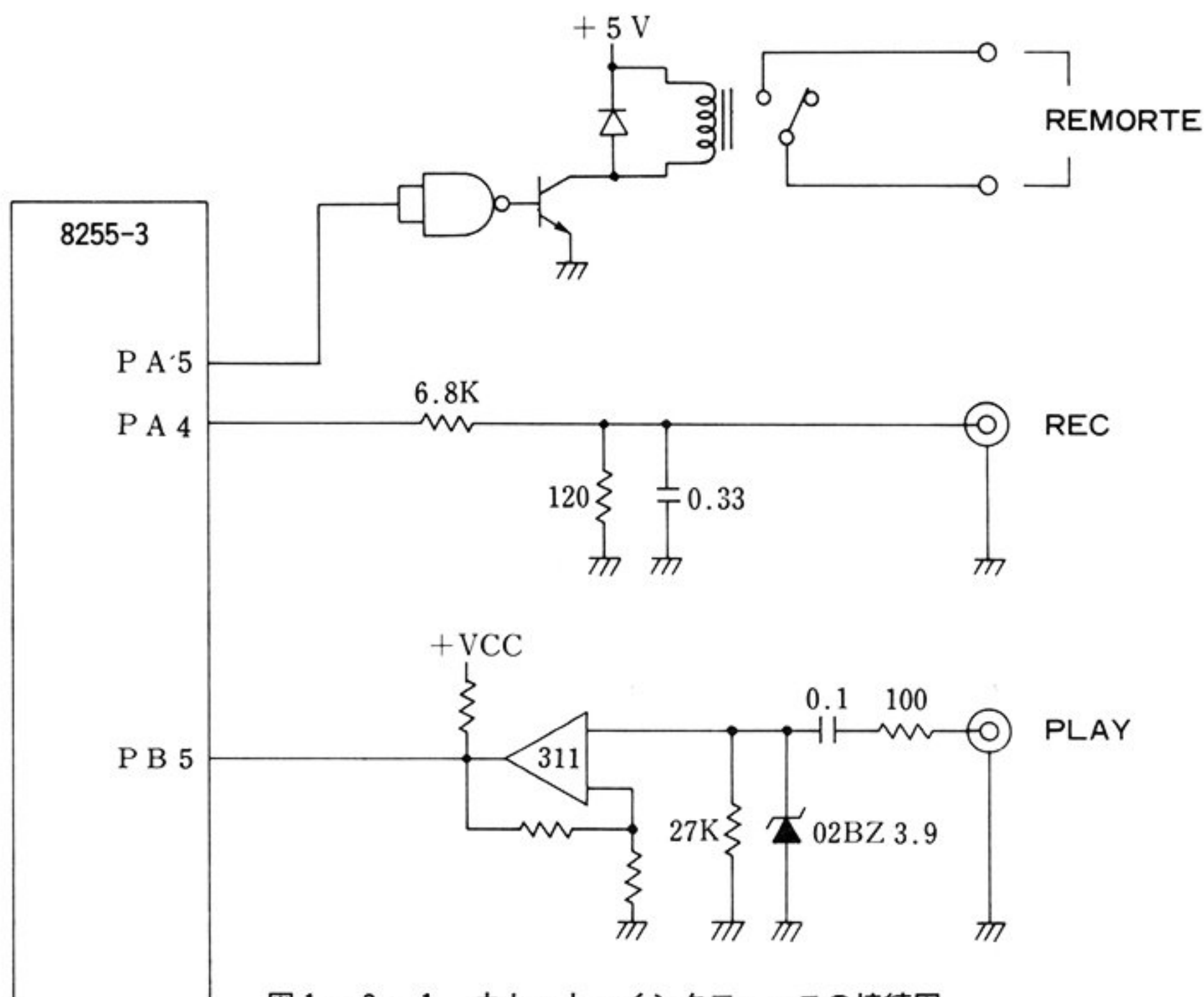



図１－３－１　カセット・インタフェースの接続図

$$\frac{1}{\dfrac{\left(\dfrac{1}{1200}+\dfrac{1}{2400}\right)}{2}}=1600\text{bps}$$

図１－３－２　ボーレートの計算式

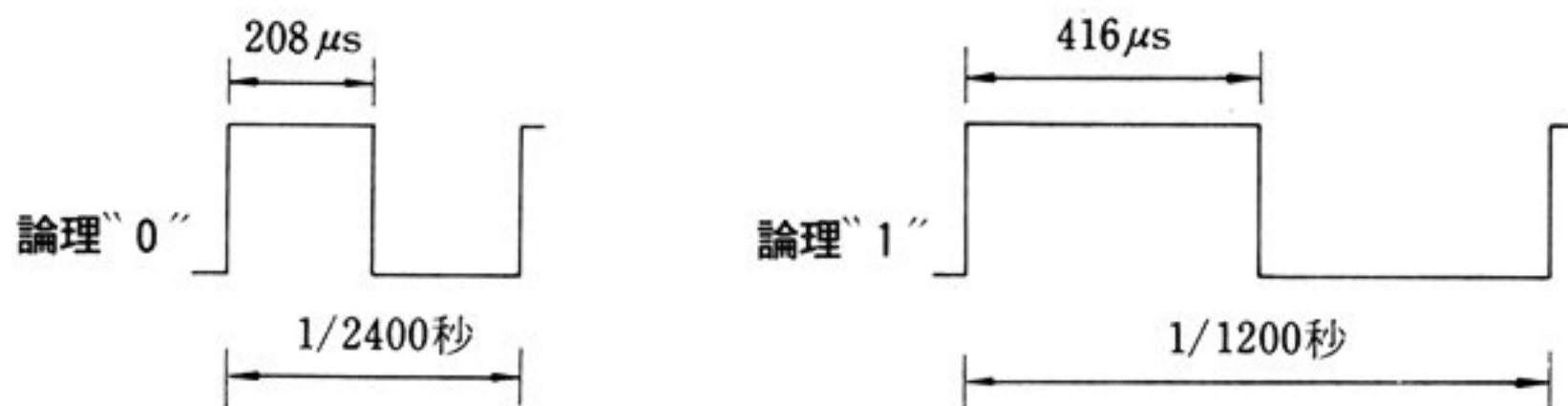



図１－３－３　データの記録波形

1-3-2　データの入出力

カセットテープへのデータ書き込みは，タイマ割込によりタイミングをとっており，１バイト分のデータの前後に同期合わせのため，スタート/ストップビットが付加されて記録されます．図1-3-4に，データの形式と割込の関係を示します．

一方，データの読み込みは，入力波形の反転(エッヂ)を検出して，エッヂから次のエッヂまでの時間によって論理"1"か"0"を判定しています．

これは，Z80CTCのカウンタ機能を利用して計測しており，エッヂから320μsの時点が，１か０

の分かれ目になっています．実際には，論理1のエッヂ間が416μs，論理0のエッヂ間が208μsとなっていますので，20%の速度変動までデータ読み込み可能ということになります．図1-3-5に，入力波形とビット判断点の図を，図1-3-6に，オーディオカセット・インタフェィスのコネクタピン配置を示します．

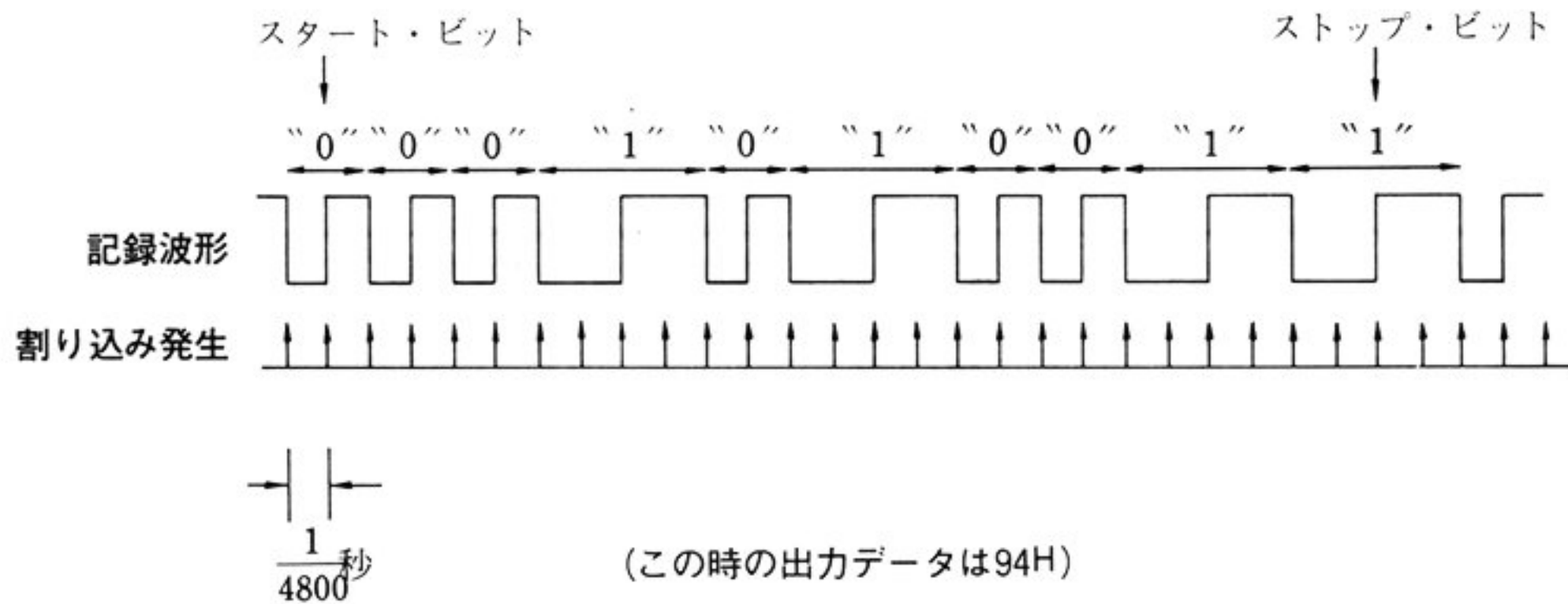



図1―3―4　データの出力形式と割り込み

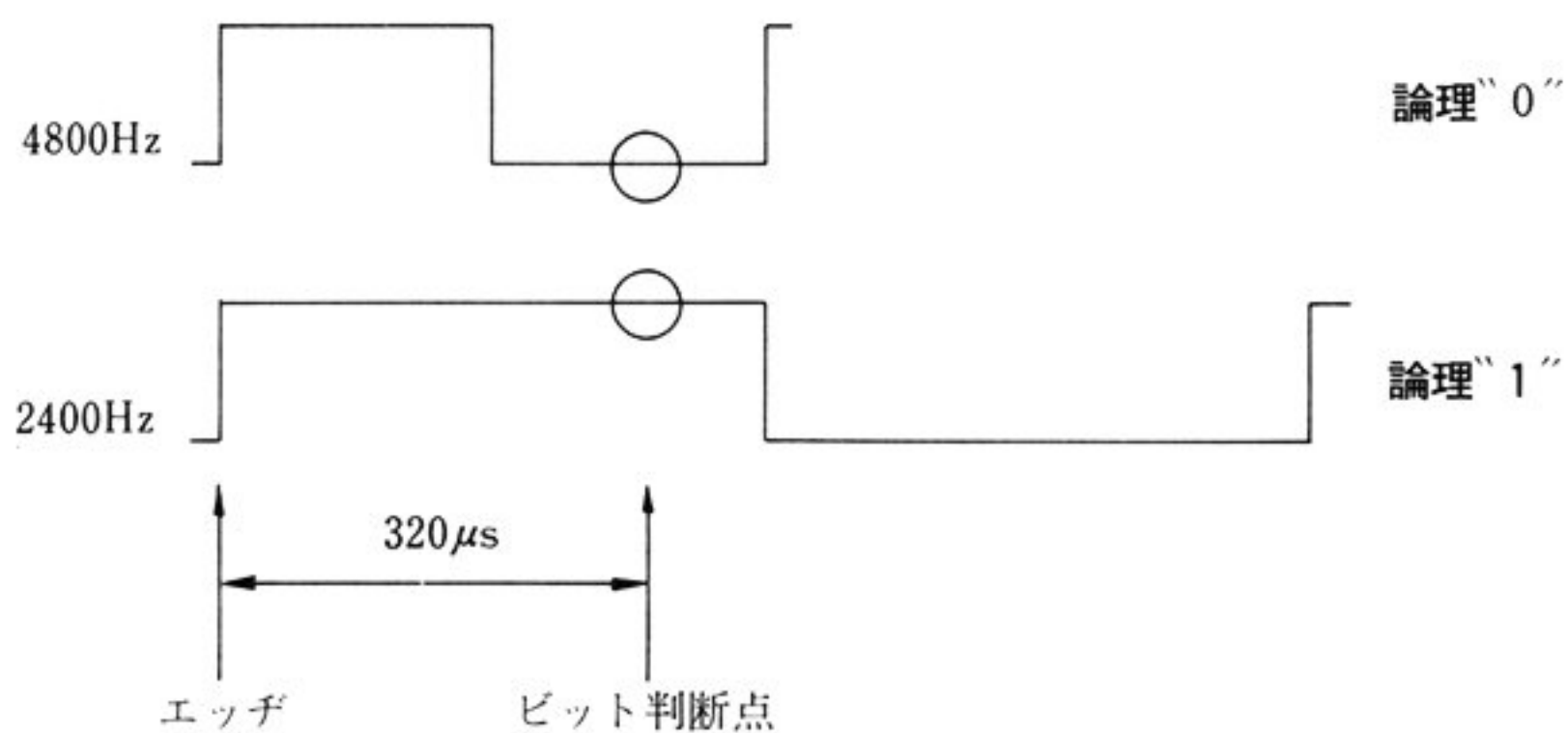



図1―3―5　入力波のビット判断

ピン番号	用　途	I／O
1	——	
2	GND	
3	——	
4	RECOUT	O
5	MONITOR	I
6	リモート＋	O
7	リモート－	O
8	GND	

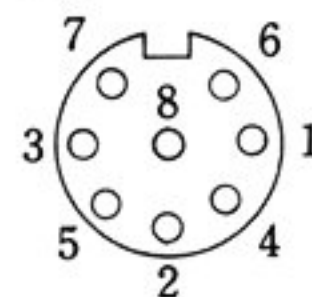



図1—3—6　オーディオカセットインターフェースコネクタ図

1-4　ディスプレイ装置とそのインタフェイス

1-4-1　CRTディスプレイ

パソピアの表示装置には，CRT(または家庭用テレビ)と液晶ディスプレイがあり，これらは専用のCRTC(HD465055)によって制御されています．図1-4-1に，表示モードと画面サイズおよび色分解能の表を示します．

表示装置	表示モード	横文字数	ドット数(ヨコ×タテ)	色分解能(ヨコ×タテ)	表示文字数
モニタテレビ または 家庭用テレビ	テキスト	80	——	80×25 (80×20)	80×25 (80×20)
		36	——	36×24 (36×19)	36×24 (36×19)
	グラフィック	80	160×100	160×100	80×25
		36	72×96	72×96	36×24
	ファイン グラフィック	80	640×200	80×200	80×24
		36	288×192	36×192	36×24
液晶ディスプレイ	テキスト	40	——	——	40×8 (40×6)
	ファイン グラフィック	40	320×64 (320　×48)	——	40×8 (40×6)

図1—4—1　各ディスプレイの表示モードと分解能

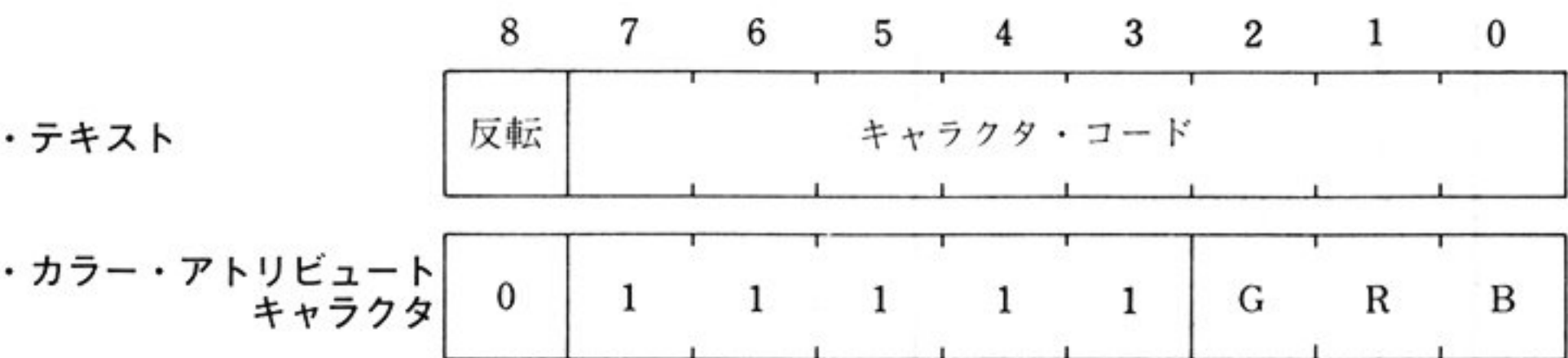
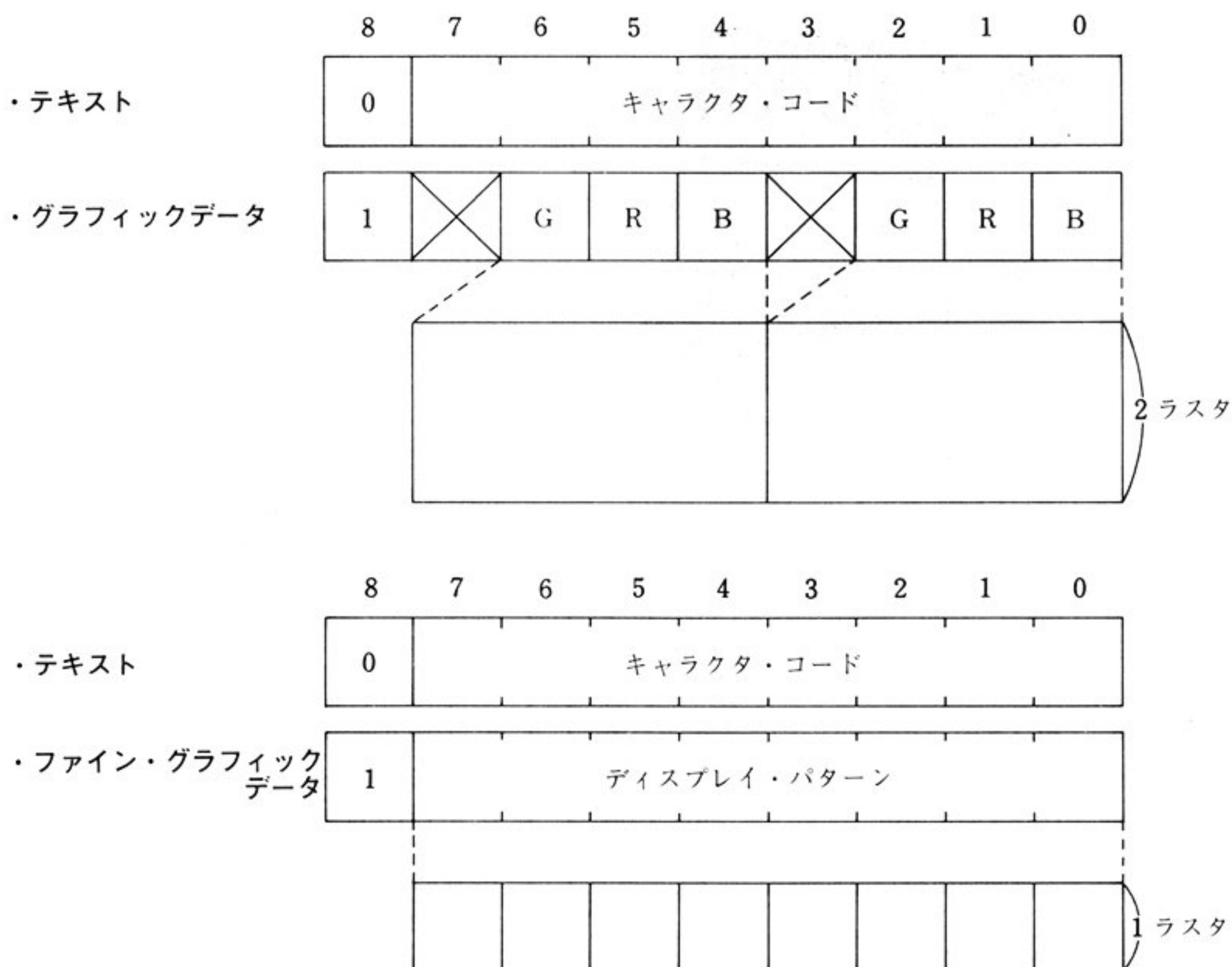
表示モードには，テキストモード，グラフィックモード，ファイングラフィックモードの3種類あり，I/Oポートの08Hにデータを出力することによって切り換えられます．図1-4-2に出力データの形式を示します．

図1—4—2　出力データの形式

	8	7	6	5	4	3	2	1	0
・テキスト	0	キャラクタ・コード							
・グラフィックデータ	1	╳	G	R	B	╳	G	R	B

2ラスタ

	8	7	6	5	4	3	2	1	0
・テキスト	0	キャラクタ・コード							
・ファイン・グラフィック データ	1	ディスプレイ・パターン							

1ラスタ

1-4-2　液晶ディスプレイ

液晶ディスプレイ装置は，パソピア本体後部に取りつけ，コネクタを本体にさし込むことによって作動し，電源も本体から供給されます(写真1-5，６).

表示画面は320×64ドットと大型で，それをドライブするための液晶ドライブ用LSIが，12個も使用されています．内部は，本体からのビデオ信号を処理する部分と液晶ドライブ用LSIから成っており，基板を３枚重ねにすることにより高密度に実装されています．(写真参照)

図1-4-3に液晶ディスプレイのブロック図とコネクタ図を示します．

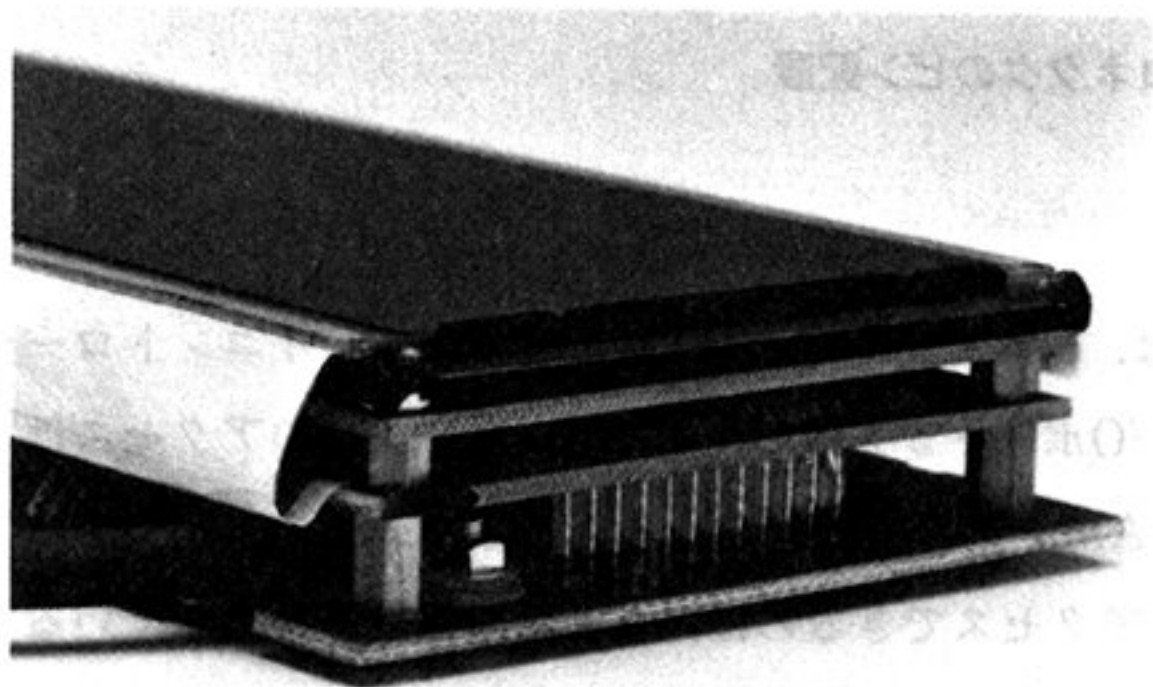
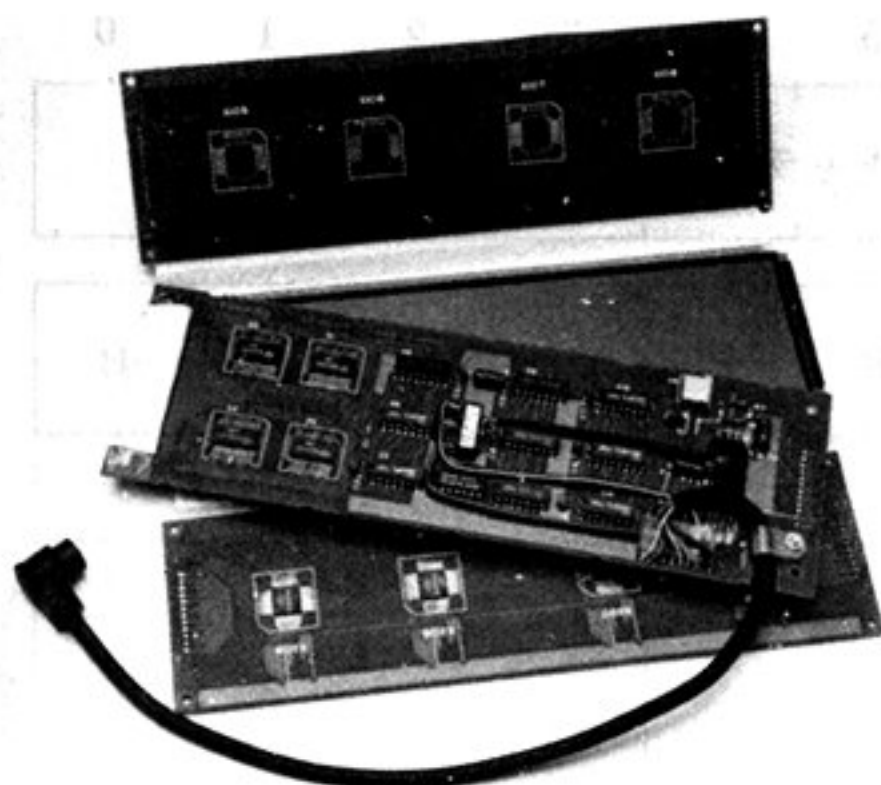



写真1-5
液晶ディスプレイ

写真Ⅰ-6
液晶ディスプレイ(内部)

P I	PIN	SIGNAL NAME	I / O	S H
13	1	GND	I / O	25
	2	NC		
	3	CRTC : 100(STATUS)	1	25
	4	BCLRBL : 010(MODE)	0	25
	5	+5V	0	25
	6	CVSYNC : 010	0	25
14	1	CVIDGR : 100(VIDEO)	0	25
	2	GND		25
	4	CHSYNC : 010	0	25
	5	$14MHZ : 000(CLOCK)	0	25
	6	−12V	0	25
	3	NC		

PIN	SIGNAL NAME	I / O	S H
1	CLOCK	O	
2	GND		
3	+5V	O	
4	MODE	O	
5	STATUS	I	
6	VIDEO	O	
7	−12V	O	
8	HSYNC	O	
9	GND		
10	VSYNC	O	

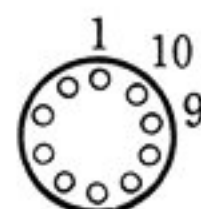
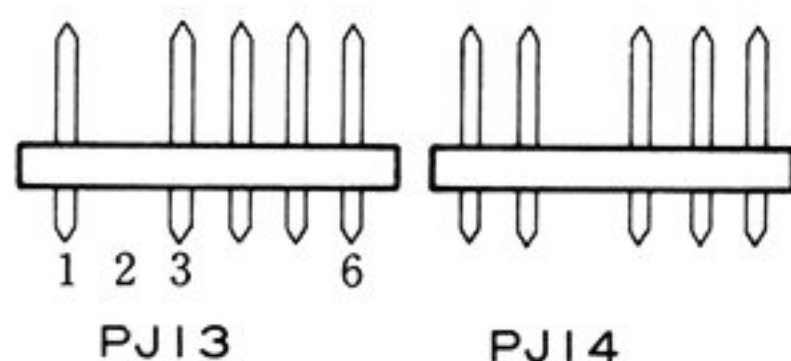



図1―4―3 LCD用コネクタのピン配置

1-4-3 ディスプレイ・インタフェイス

パソピアのディスプレイ・インタフェイスは，9bit×16KのV-RAMとディスプレイコントローラ(CRTC)そして，それらを制御するためのI/Oポートから成っています．V-RAMのアクセスはI/Oポート経由で行えますが，CRTCが，V-RAMをアクセスしているときは，CRTの方が優先されます．そのため，I/OポートからV-RAMをアクセスできるのは，水平同期信号を出力しているときだけになります(図1-4-4参照)．

グラフィック用I/Oポートは，PPI(8255)を2個使用することによって6つ設けられています．

内わけは，９ビットのグラフィックデータのリードとライトを別々のI/Oポートで転送しているため，14ビットのアドレスデータと合わせて32ビットが必要となり，８ビットのI/Oポートが全部で４つ使われます．また，残りの２つのI/Oポートは，表示モードの切り換え，背景色の設定，ステータスリード等に使用されています．図１-４-５に，I/Oポートまわりの接続図を示します．

の部分は、CRTCがV-RAMを専有しているので、アクセスできない。

図１－４－４　V-RAMアクセスタイム

図１－４－５　I/Oポートの接続図

V-RAMの読み出しと書き込みは完全に独立しており，データの書き込みはI/Oポートからのみ行われます．データの読み出しは，V-RAMからのデータが二又に別れた後それぞれラッチされ，1つはI/Oポートに行き，CPUからのデータの読み出し用に使われます．もう1つは，グラフィックデータか，カラーアトリビュート・キャラクタか，キャラクタコードかによって別れ，キャラクタコードはキャラクタジェネレータ(2716)によってキャラクタパターンに変換され，出力されます．図1-4-6にV-RAM周辺の構成図を示します．

図1—4—6　V-RAM周辺の構成図

1-5　ミニフロッピィ・ディスクユニット

1-5-1　概要

ミニフロッピィ・ディスクユニットは，パソピア本体の外部バスに接続され，最大4ドライブまで拡張することができます．ドライブは両面倍密度形で，約285KBの記憶容量を持っており，専用のFDC(μPD765)によって制御されています．

このドライブは動作が非常に静かで，ヘッドがディスクをアクセスしたままのように見えますが，実はヘッドとディスクの間が5ミクロンほど離れており，その状態でREAD/WRITEが行われています．これはヘッドがごく弱い力で，ディスクをアクセスするため，ディスクの回転によって起こる空気の流れで，ほんの少しヘッドが浮くという東芝独自のフローティング構造になっているためです(写真1-7)．

図1-5-1にミニディスクユニットPA7200の仕様を示します．

写真1-7
ミニフロッピイディスク(フローティングユット)

データ転送速度	250キロビット／秒	
平均アクセスタイム シークタイム セットリングタイム 平均回転待ち時間	398ミリ秒 25ミリ秒／トラック 15ミリ秒 100ミリ秒	
駆動モーター起動待ち時間	1.5秒	
回転速度	300回転／分	
ミニフロッピーディスク装置台数	最大4台	
記録密度	両面倍密度(直径5.25インチ)	
記録容量 　ディスク1枚 　1トラック 　1セクタ	 280KB(1KB=1024) 4096バイト 256バイト	
トラック数	70(片面35トラック)	
トラック当りセクタ数	16	
電源電圧	AC100V±10%　50／60Hz	
消費電力	約70W	
寸法	420(幅)×270(奥行)×130(高さ)mm	
重量	約9kg	
	温　度	湿　度
動作時	5～35℃	20～80%
非動作時	−15～47℃	10～90%

図1－5－1　ミニフロッピィディスク・ユニットの仕様

1-5-2　ミニディスクユニットの構成

ミニフロッピィ・ディスクユニットPA7200は，2台のドライブとFDC基板から成っており，インテリジェント化はされておりません．このため，FDCはパソピア本体から直接制御され，クロックも本体から供給されています．

また，パソコン用のミニディスクユニットにはめずらしく，放熱用のファンが取り付けられ，信頼性と耐久性が一段と向上しています．図1-5-2に，ブロック構成図を示します．

1-5-3　データの転送とタイマの補正

パソピア本体とCPUとのデータ転送は，0トラックの0サイドを除いてはすべて256バイト単位で行われます．それはCPUが直結されているため，バッファがFDC内の256バイトだけだからです．

ディスクをアクセス中は割込を禁止しています．このため，アクセス終了後，システムタイマを補正する必要があります．実際には，1/64秒の割込が入った直後，割込を禁止し，図1-5-3のように動作完了タイミングにより，システムタイマの補正を行っています．

図1―5―3　システムタイマ補正

写真 I - 8
ミニフロッピイディスク(内部基盤)

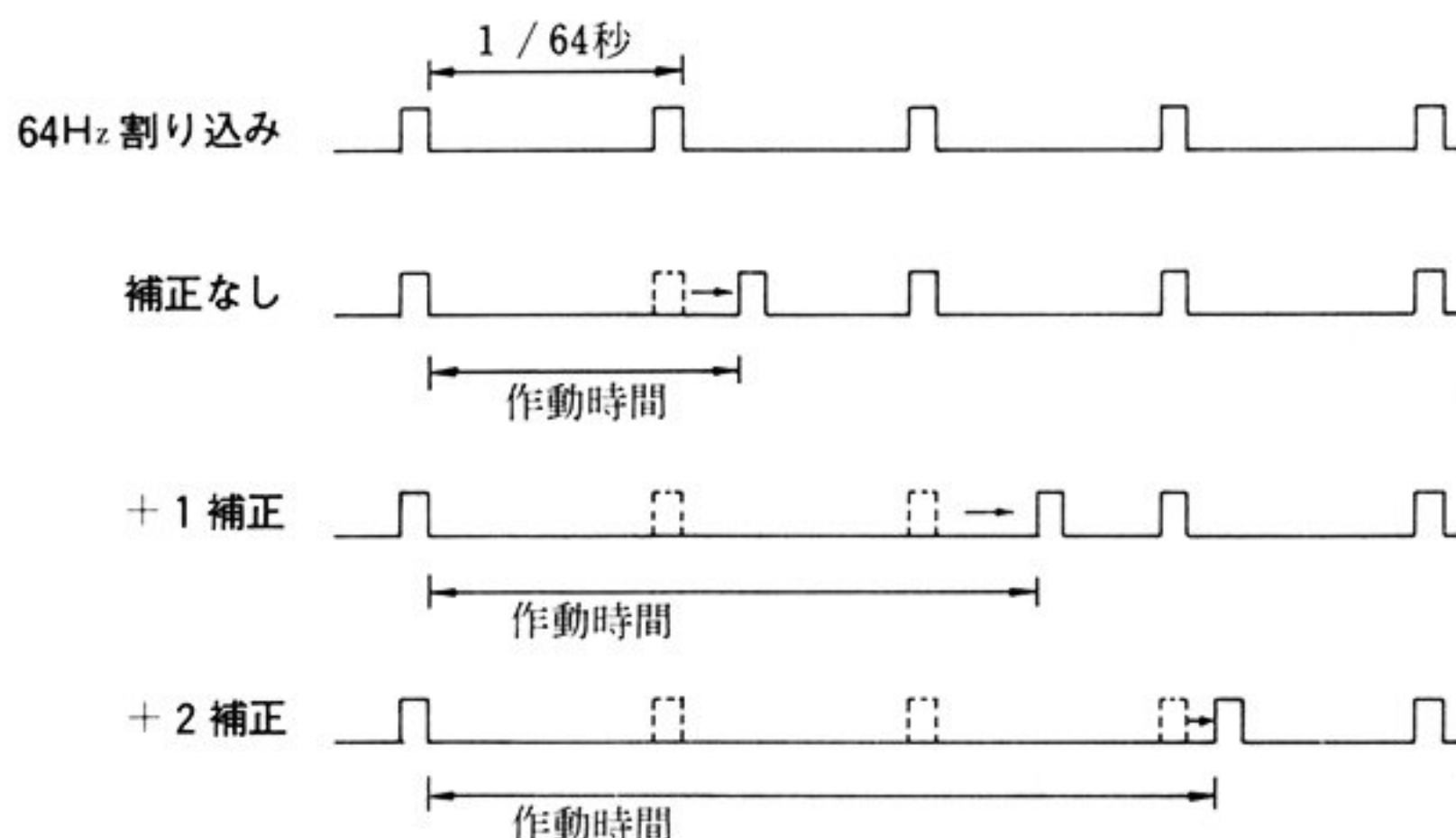
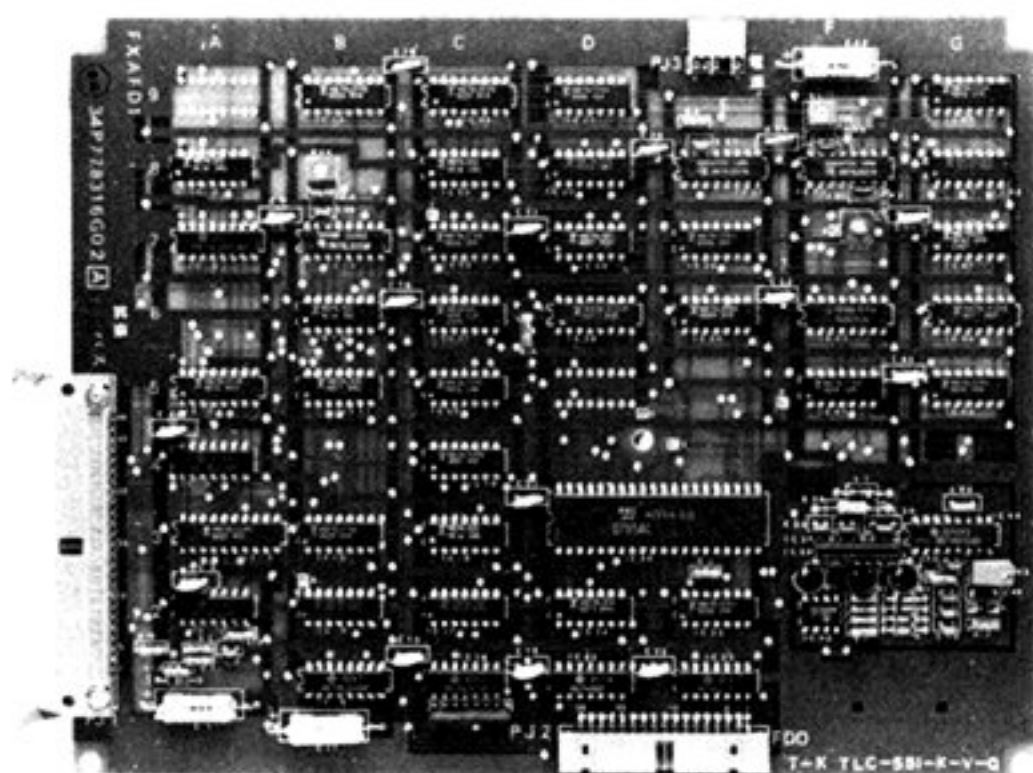
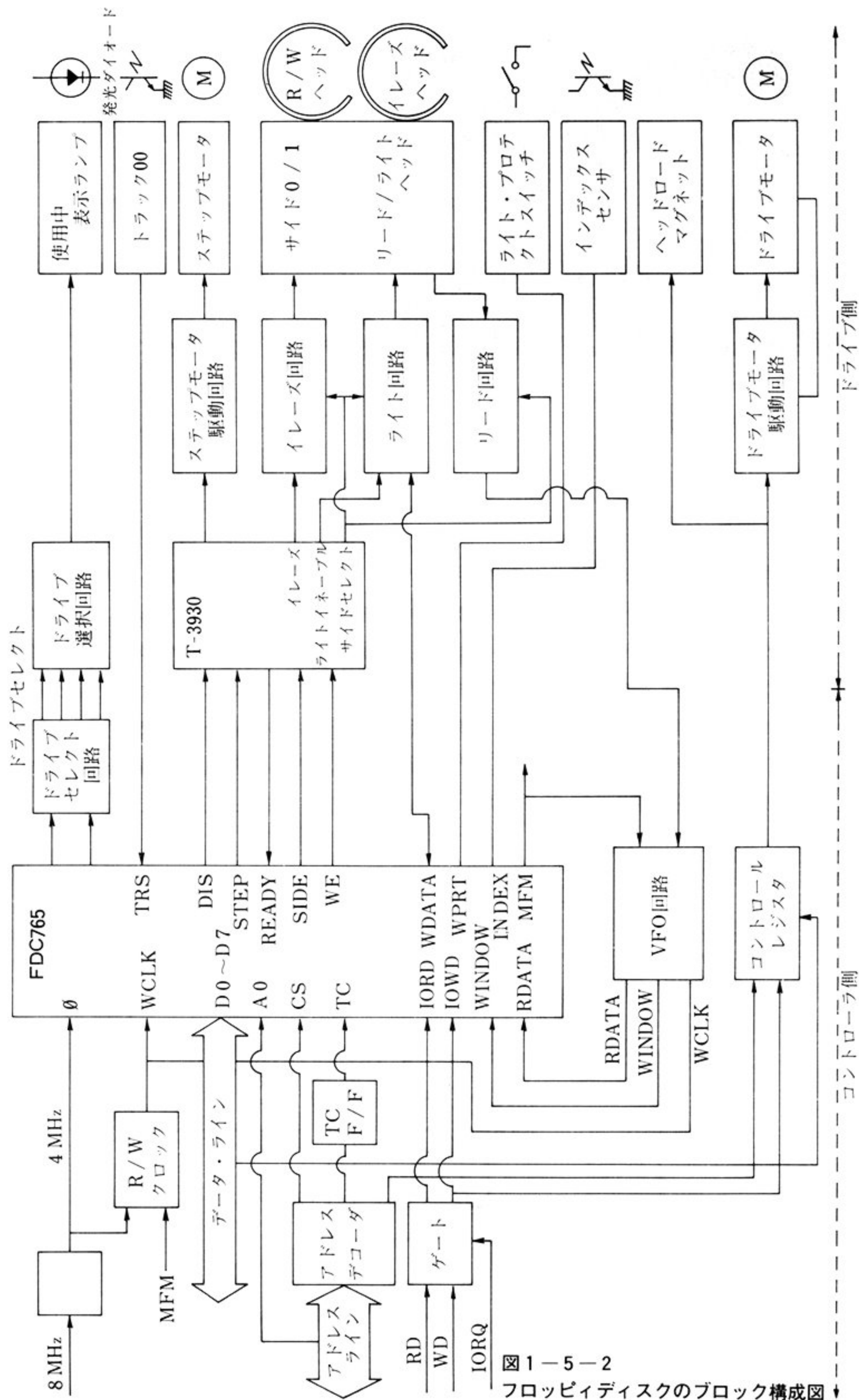



図1－5－2
フロッピィディスクのブロック構成図

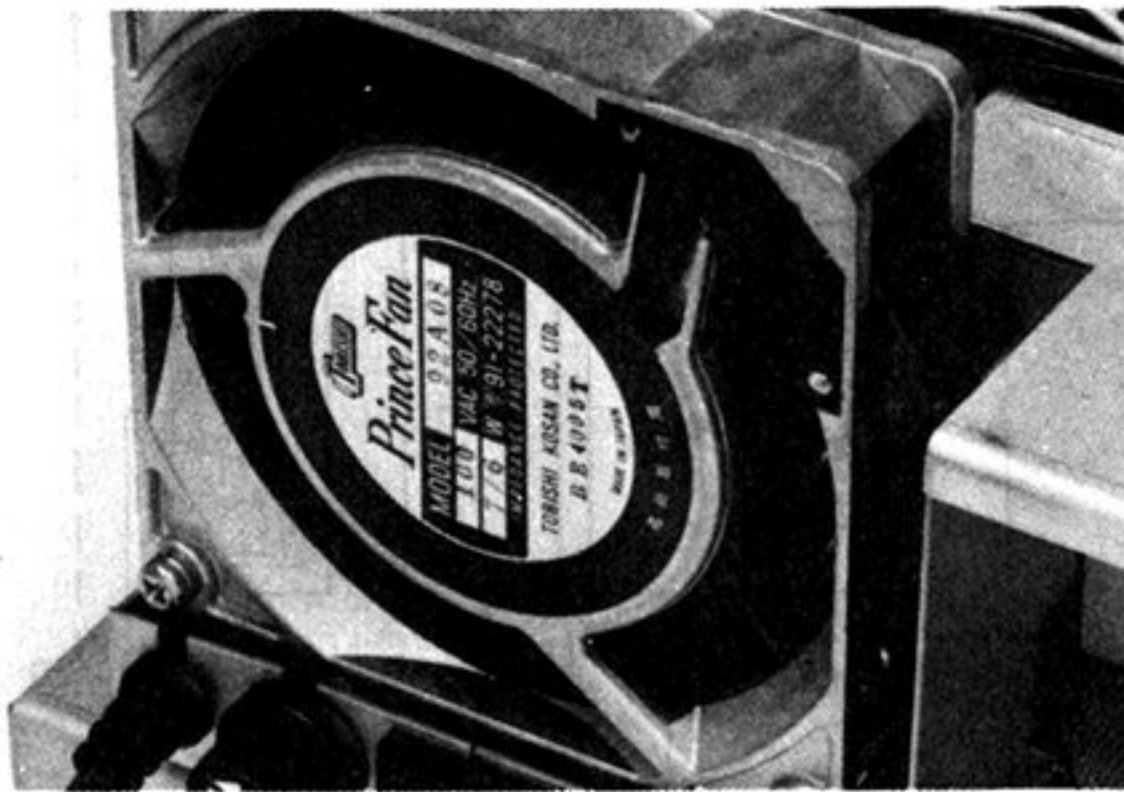



写真1-9
ミニフロッピイディスク(ファン)

1-6 プリンタ

1-6-1 概要

パソピア専用のプリンタは，低価格のドットプリンタI(PA7250)と，高品質，多機能で高速印字ができるドットプリンタII(PA7251)の2機種が用意されています(写真1-10)．また，プリンタ・インタフェィスはセントロニクス準拠となっているため，他社のプリンタ(例えば，エプソンのMP-80など)も接続することができます．プリンタ・インタフェィスは，本体のPPI(8255-3)によってコントロール信号の制御を行い，データは，データバスからバッファ(74LS373)を介して出力されます．データ転送のタイミングは，STROBEとBUSYでとっています．図1-6-1に各ドットプリンタの仕様を，図1-6-2にプリンタ・インタフェィスの接続図を示します．

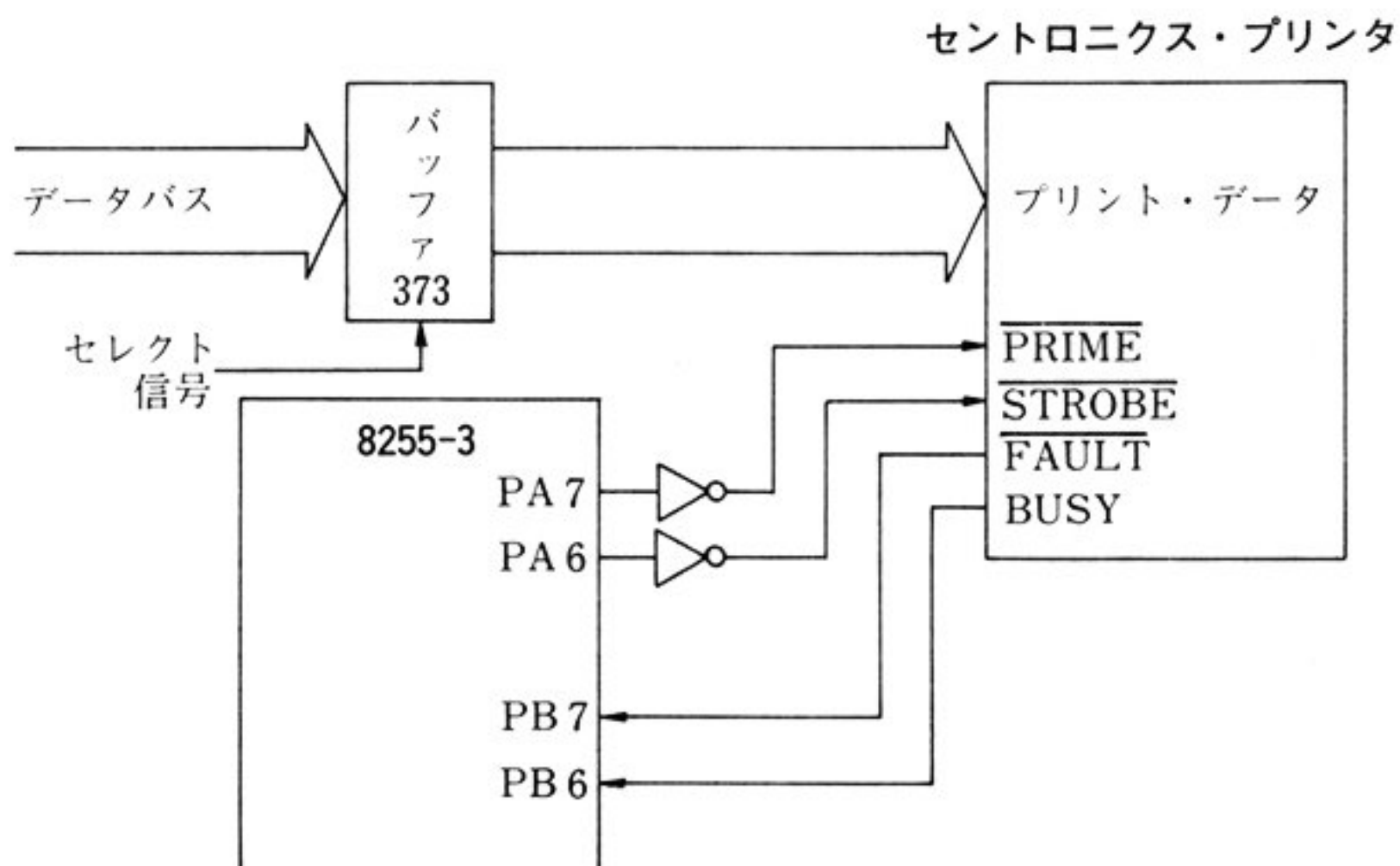
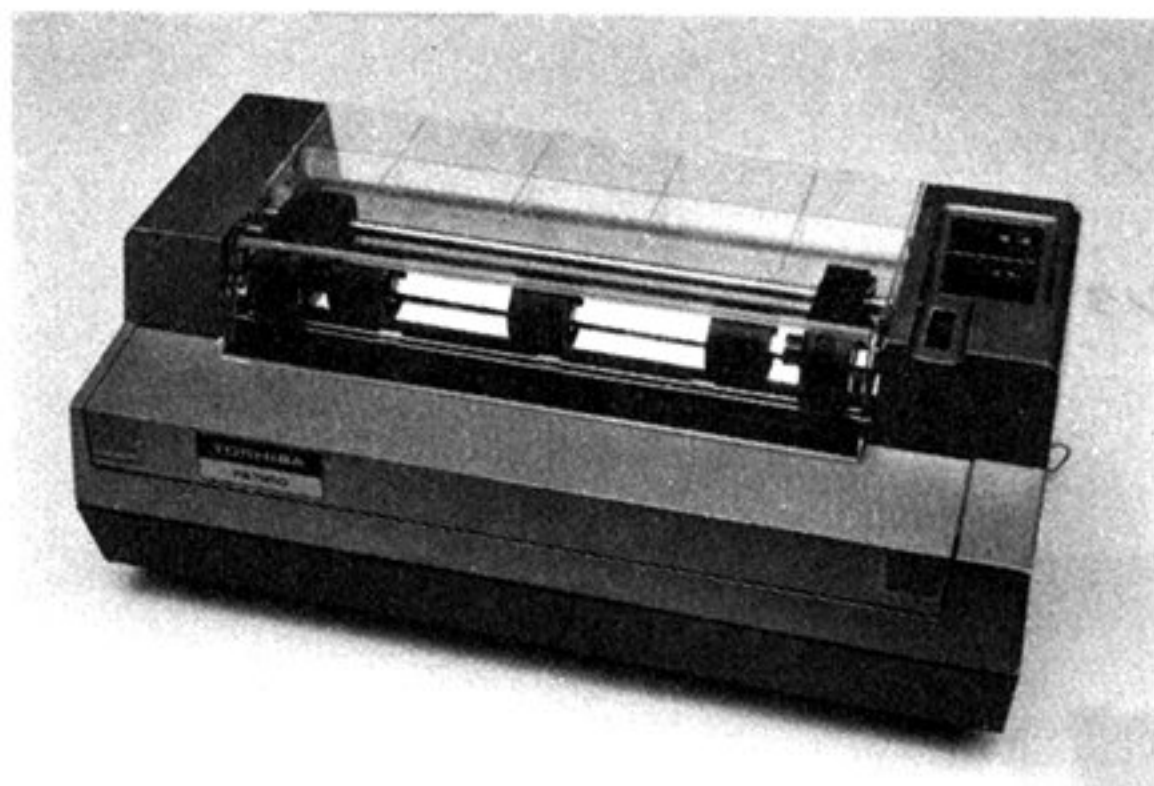



図1―6―2 プリンタ・インタフェースの接続図

	ドット・プリンタⅠ PA 7250	ドットプリンタⅡ PA 7251
印字方式	ドットインパクト 片方向	ドットインパクト 両方向最短距離
印字速度	30字／秒	120字／秒
ドット構成	5×7	8×9
印字桁数	80字／行	80字／行
印字文字種	JIS160種	JIS 160種 プロポーショアル　35種 ひらがな　64種 キャラシェネ・グラフィック64種
グラフィック	可(ドット対応)	可(ドット対応)
その他	倍幅印字 対応)	倍幅印字 縮小印字 1インチカット可
用紙	8インチまでのファンホールド紙	10インチまでのファンホールド紙 単票

図1－6－1　各プリンタの仕様

1-6-2　ドットプリンタⅠ

写真Ⅰ-10
プリンタⅠ

ドットプリンタⅠは，ビットイメージ・プリントができる，小型低価格のドットプリンタです．印字速度は30文字/秒と比較的低速ですが，ホビー用としては十分な性能を持っています．規格や性能は，精工舎のGP-80を流用しているため，それと同等と考えてよいでしょう．図1-6-3にタイミングチャートを，次ページ図1-6-4にブロック図を示します．

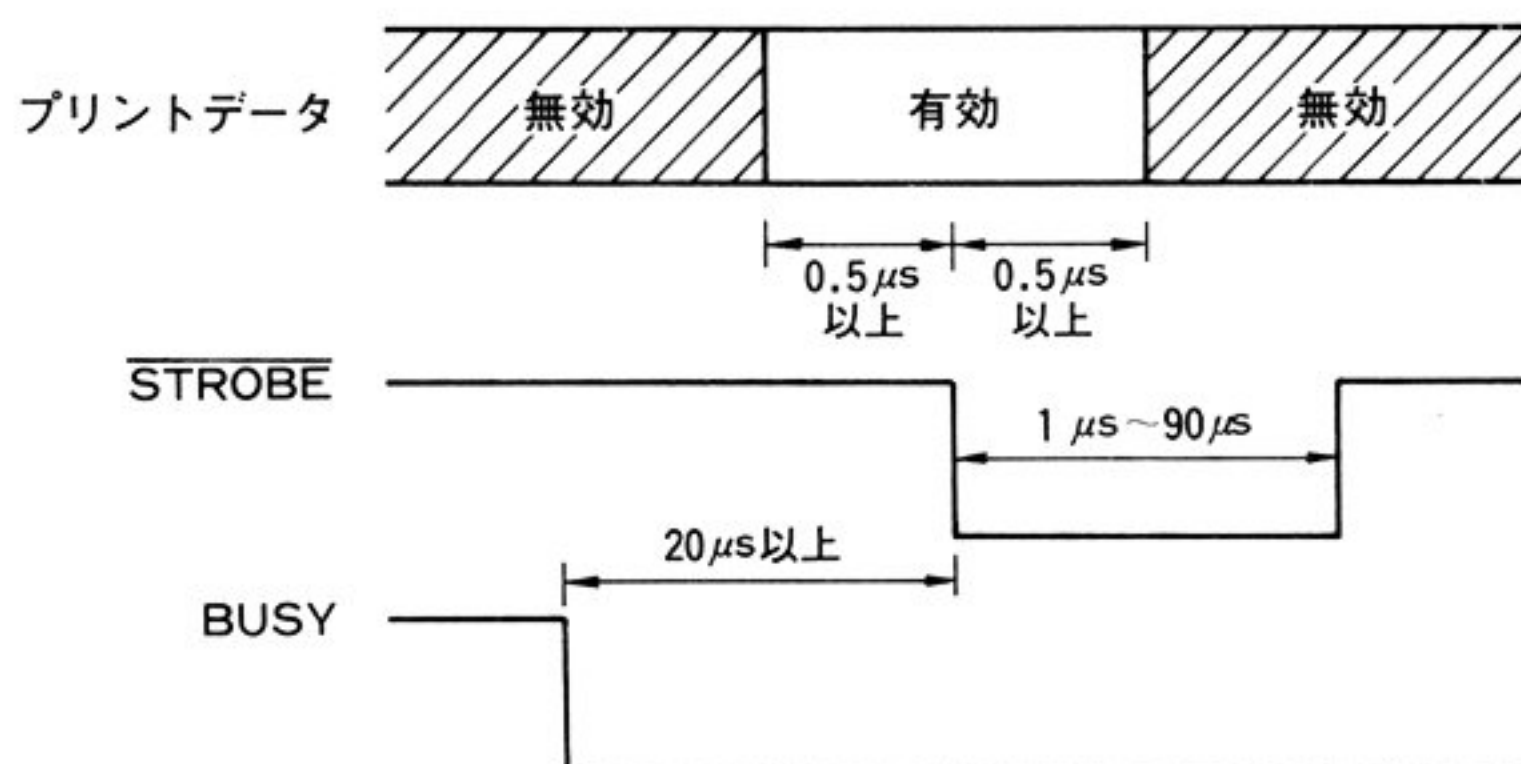



図1―6―3　プリンタⅠのタイミングチャート

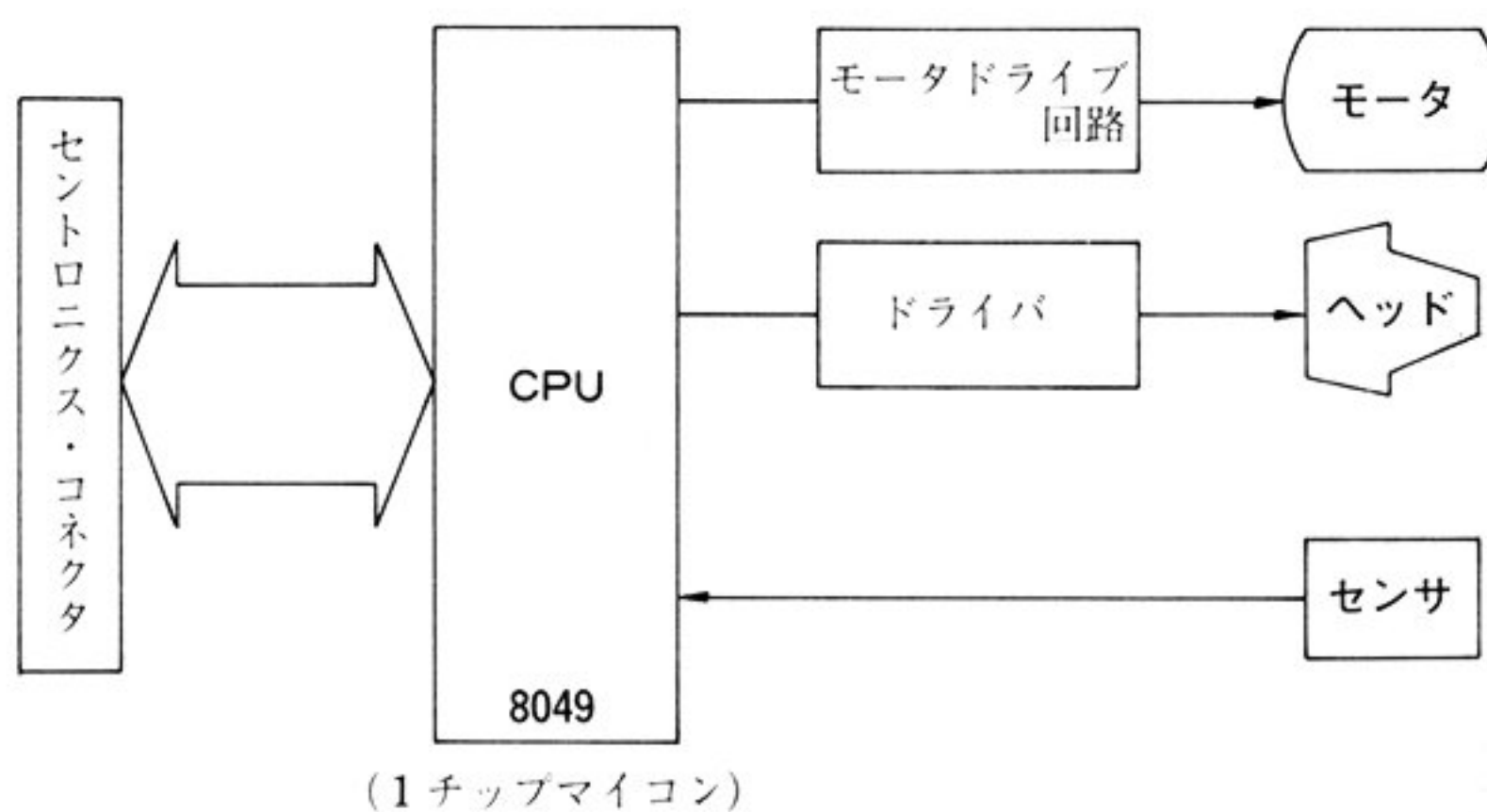



図1―6―4　プリンタⅠのブロック図

1-6-3　ドットプリンタII

写真Ⅰ-11
プリンタII(外観)

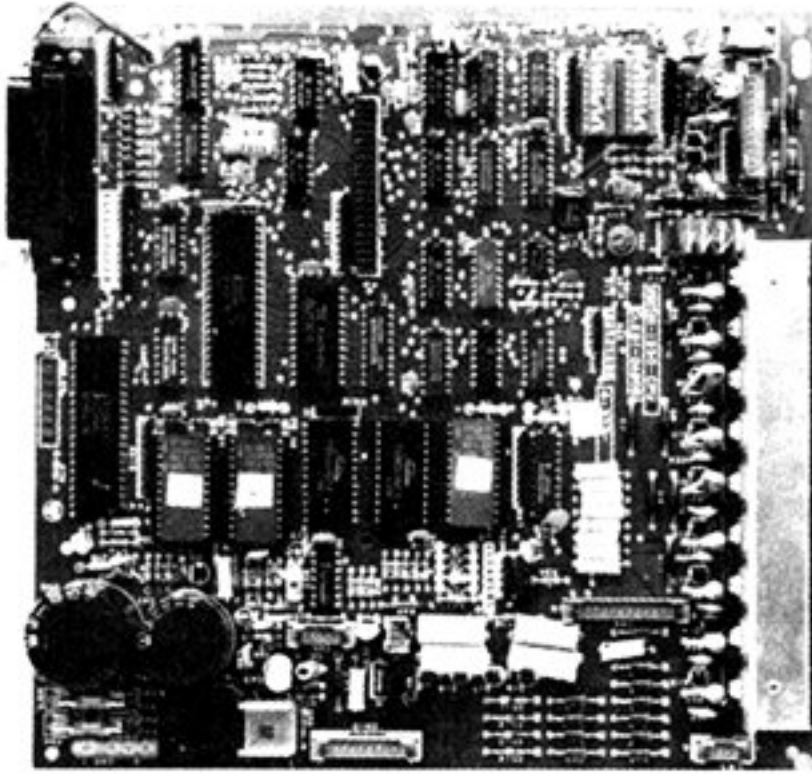



写真1-12
プリンタII(内部基盤)

ドットプリンタIIは，ビジネスニーズに対応できる品質，高速印字を特徴としています．このプリンタはインテリジェントタイプで，プリンタの制御をCPU(8085)で行っています．また，約2Kバイトのデータバッファを持っているため，少量印字のときは大幅にCPUの待ち時間を節約することができます．図1-6-5にブロックを，図1-6-6にタイミングチャートを示します．

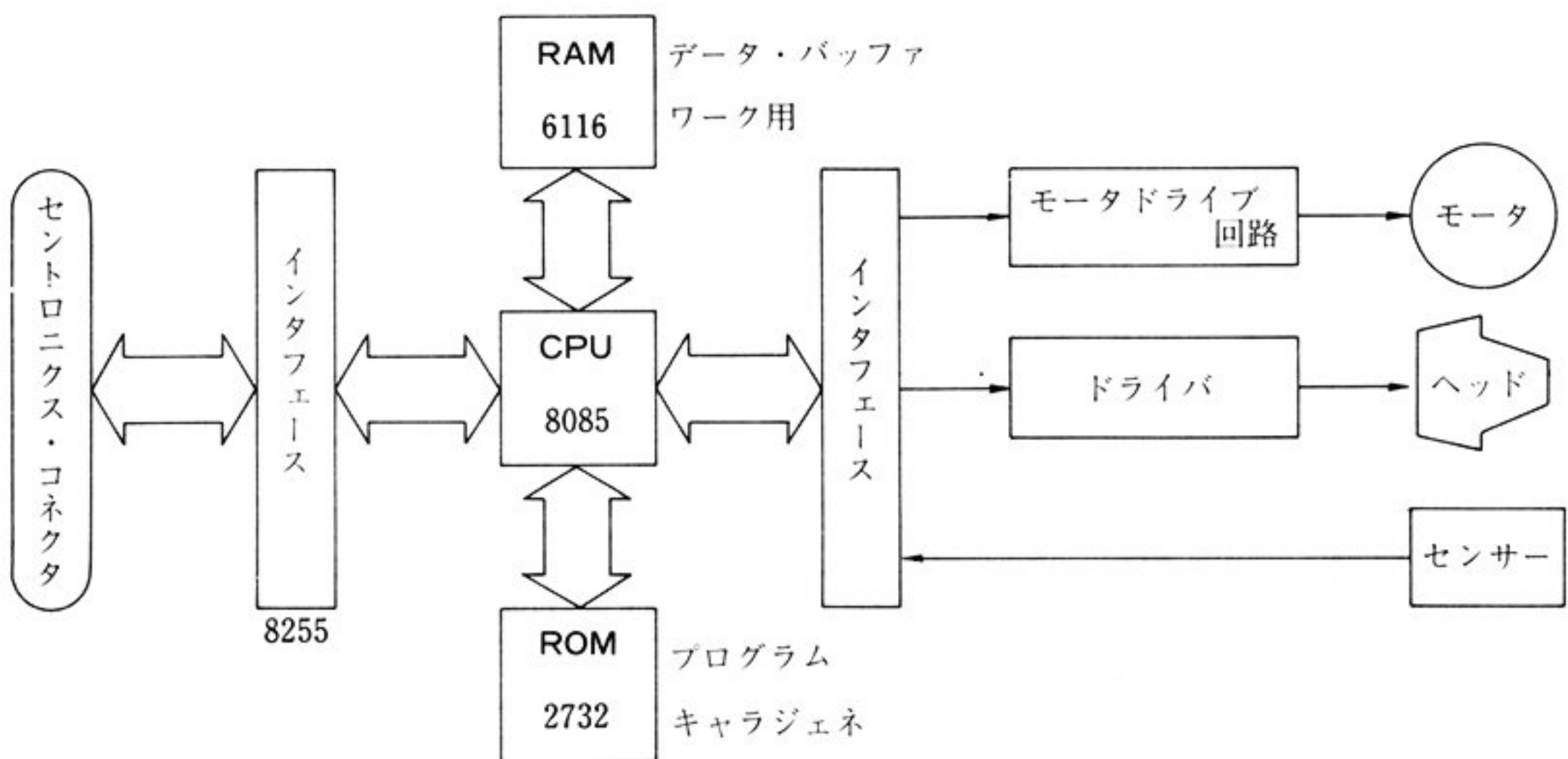



図1―6―5　プリンタIIのブロック図

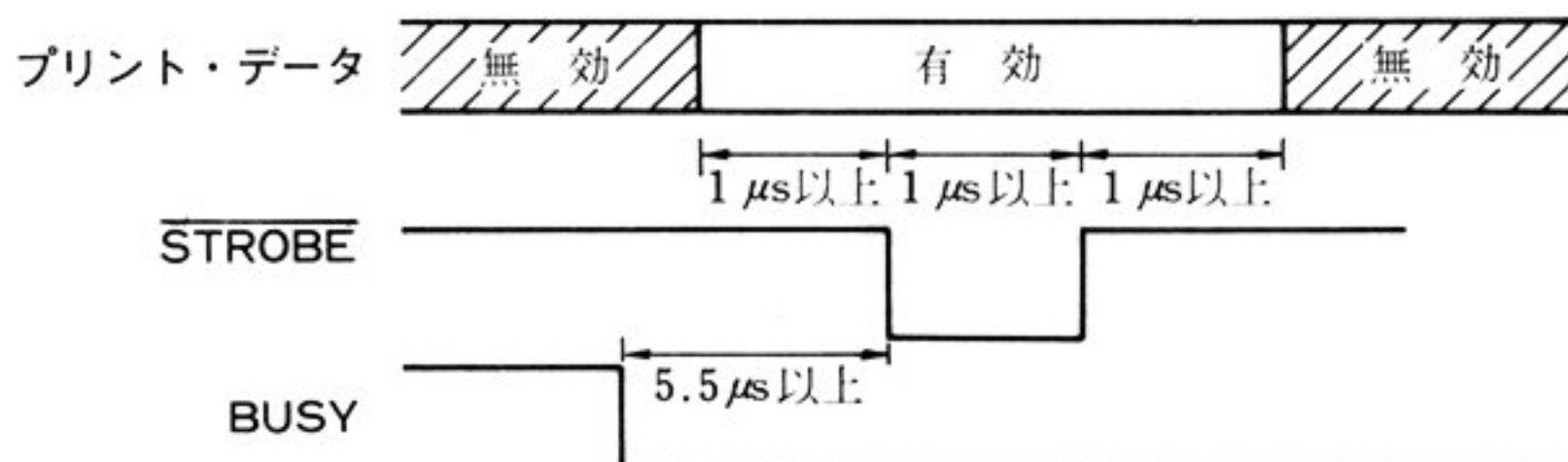



図1―6―6　プリンタIIのタイミングチャート

また，信号ラインを図1-6-7に示します．

ピン番号	信 号 名	I／O	ピン番号	信 号 名	I／O
1	STRB	O	2	GND	
3	PDATA 0	O	4	GND	
5	PDATA 1	O	6	GND	
7	PDATA 2	O	8	GND	
9	PDATA 3	O	10	GND	
11	PDATA 4	O	12	GND	
13	PDATA 5	O	14	GND	
15	PDATA 6	O	16	GND	
17	PDATA 7	O	18	GND	
19	$\overline{\text{PRIM}}$	O	20	GND	
21	PBUSY	I	22	GND	
23	$\overline{\text{FAULT}}$	I	24	GND	
25	OSCXT	O	26	PHLVL	O

プリンタ　結線図　コネクタ（本体内PJ9）

ピン番号	信 号 名	I／O	ピン番号	信 号 名	I／O
1	STRB	O	2	GND	
3	PDATA0	O	4	GND	
5	PDATA1	O	6	GND	
7	PDATA2	O	8	GND	
9	PDATA3	O	10	GND	
11	PDATA4	O	12	GND	
13	PDATA5	O	14	GND	
15	PDATA6	O	16	GND	

（本体内PJ 10）

ピン番号	信 号 名	I／O	ピン番号	信 号 名	I／O
1	PDATA7	O	2	GND	
3			4	GND	
5	PBUSY	I	6	GND	
7			8	GND	
9			10	$\overline{\text{PRIM}}$	O
11			12	$\overline{\text{FAULT}}$	I
13	OSCXT	O	14		
15			16	PHLVL	O

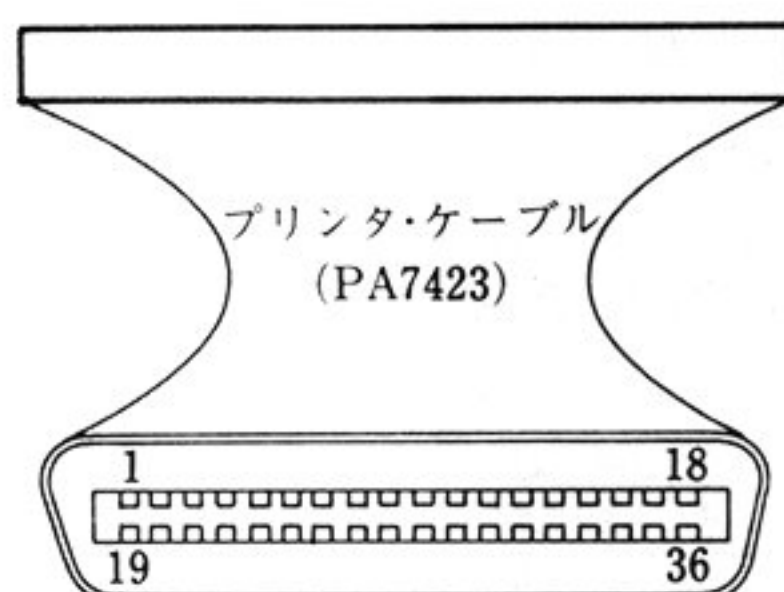
↑ O：出力
I：入力

ピン番号	信　号　名	ピン番号	信　号　名
1	STRB	19	GND
2	PDATA0	20	GND
3	PDATA1	21	GND
4	PDATA2	22	GND
5	PDATA3	23	GND
6	PDATA4	24	GND
7	PDATA5	25	GND
8	PDATA6	26	GND
9	PDATA7	27	GND
10		28	GND
11	PBUSY	29	GND
12		30	GND
13		31	$\overline{\text{PRIM}}$
14		32	$\overline{\text{FAULT}}$
15	OSCXT	33	
16		34	PHLVL
17	FG	35	
18		36	

図1－6－7　プリンタ用のコネクタ図

1-7　ROM／RAMカートリッジ

1-7-1　概要

ROM/RAMカートリッジには2つのタイプがあり，1つはバンク切換によってメインメモリの下位32Kバイトに割り当てられ，CPUから直接アクセスされます．これはROMカートリッジのみで，ROMPACと呼ばれ本体のスロットの上段に装着されます．

もう1つは，I/Oポートの経由でアクセスされ，ソフトウェアからは一種のファイルとして扱われます．こちらは，ROM，RAMカートリッジのいずれも使用でき，ROM/RAMPAC-2と呼ばれ，スロットの下段に装置されます．ROM/RAMカートリッジ自体は，プラスチック製のケースに納められていて，コネクタの部分は内側に入り込んでいるため，取り扱いにはあまり気を配る必要がありません(写真1-13)．

写真 1 -13
外観

1 - 7 - 2　ROMPAC

ROMPAC-1の最大容量は32Kバイトで，バンク切換によってメインメモリ上におかれるため，各種言語，OSなどを入れて使用することができます．その接続図を，図 1 - 7 - 1 に示します．

現在のところ，T-BASIC・Ver1.1(PA-7520)，OA-BASIC(PA-7522)，MINI-PASCAL(PA-7540)がサポートされています．このうち，PASCALの内部基盤を写真 1 -14に示します．

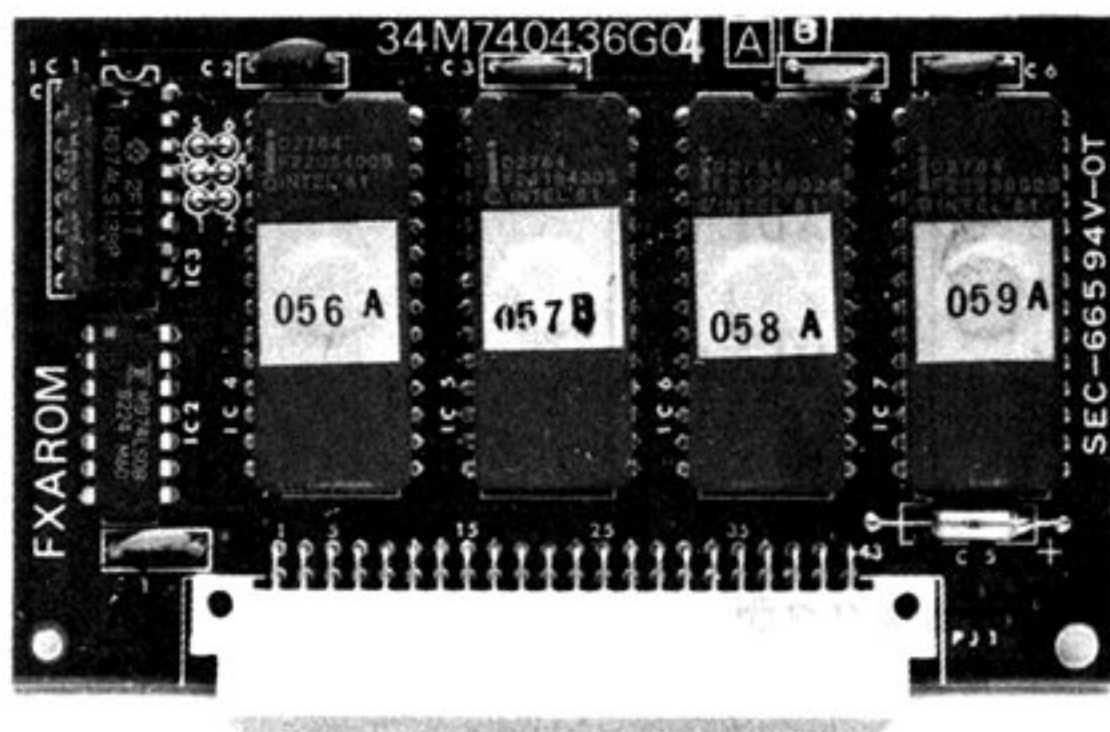



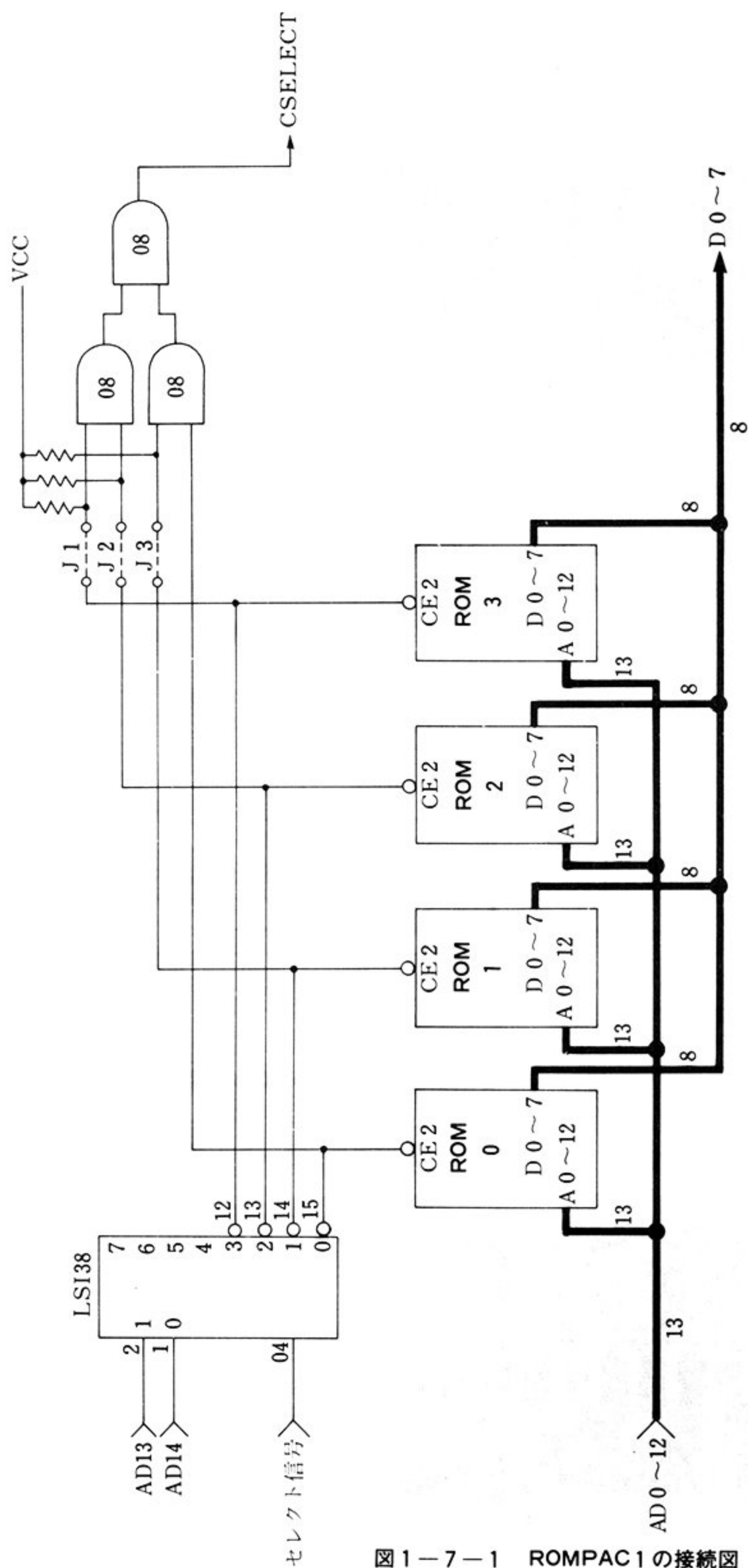
写真 1 -14
MINI-PASCAL

1 - 7 - 3　ROM／RAMPAC- 2

ROM/RAMPAC-2は，I/Oポートを介してCPUとデータのやりとりをしますが，本体にはI/Oポートはなく，データバスと，アドレスバスの 0 ～ 1 と，READ/WRITE，そしてセレクト信号が出ているだけです．このため，ROM/RAMPAC内にI/Oポートが設けられています．

アドレスバスが 2 本出力されているので，8 ビットのポートを 4 つまで設けることができます．このことは，1 つのポートをデータ入出力用に使用すると，残りの 3 つのポートがアドレス指定に使用できるので，24ビット分，つまり約16Mバイトまでアクセス可能であることになります．また，ROMの場合はデータが入力方向だけなので，4 つのポートをすべてアドレス指定に使用できます．この場合，実に4294Mバイトまでアクセスすることが可能になっています．

現在のところ，4KRAMPAC(PA-7240)，16KRAMPAC(PA-7242)，漢字ROMPAC-2(PA-

図1－7－1　ROMPAC 1 の接続図

7246)がサポートされています.

・ 4，16KRAMPAC(PA-7240)

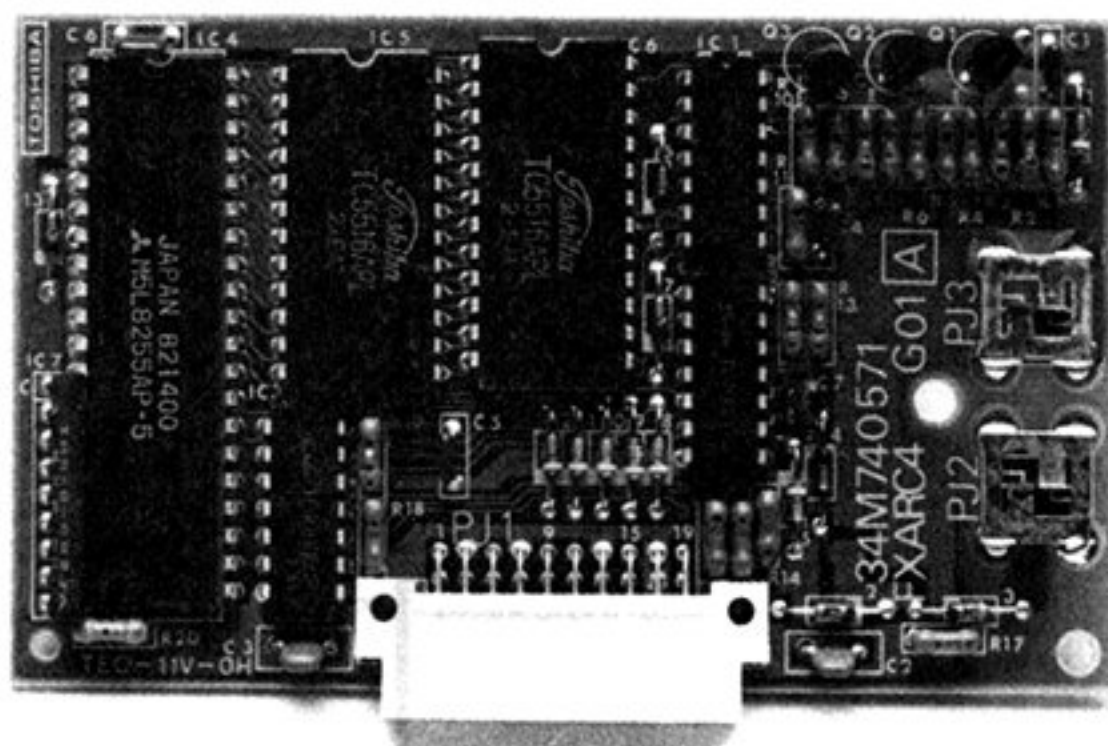



写真 1-15
4KRAMPAC

4KRAMPAC(PA-7240)は，16KビットのCMOS･RAM(6116)が2個実装されており，電池によってバックアップされているため不揮発性メモリとし扱うことができます.I/Oポートは8255(PPI)を使用しており，3つのポートのうち2つをアドレス指定に使用し，残りの1つは使用していません．図1-7-2にブロック図を示します.

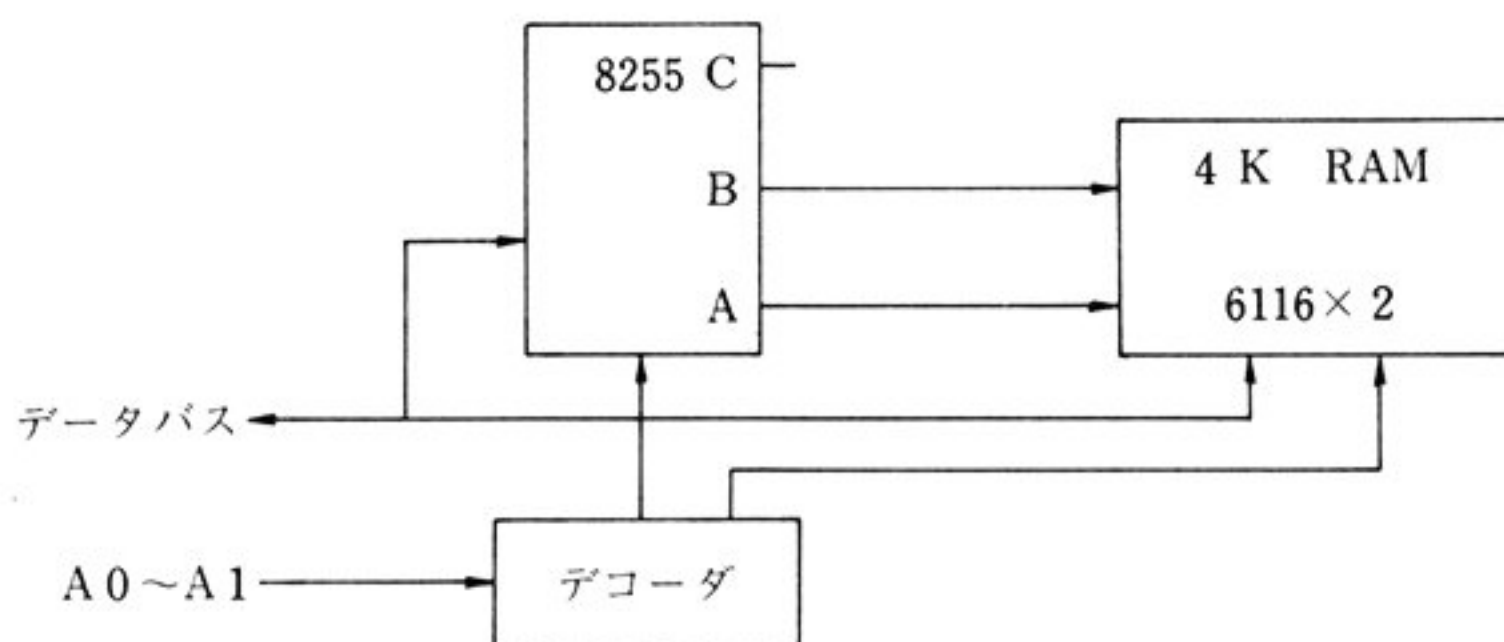



図1－7－2　4KRAMPACのブロック図

16KRAMPACは16KビットCMOS-RAMを8個使用しています.

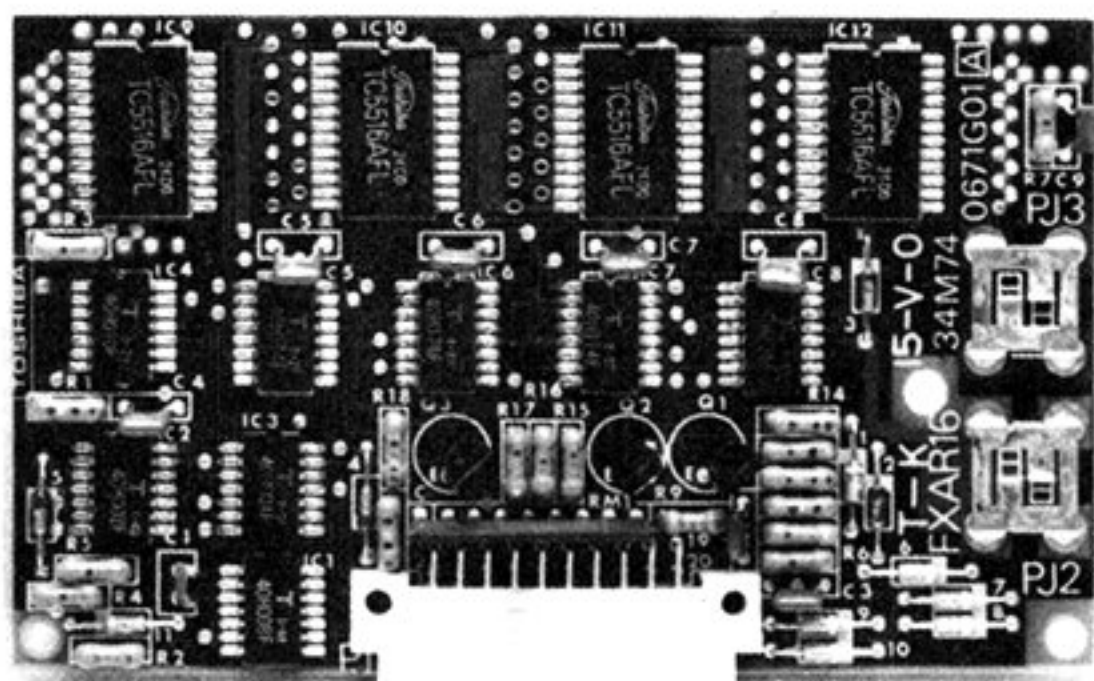



写真 1-16
16KRAMPAC

・ROMPAC-2

ROMPAC-2は，ROMを3個まで実装でき，23256をはじめ，2364や2732などの各種ROM・PROMを使用することができます．メモリ容量は，23256を使用した場合，96Kバイトで2364，2732の場合は，それぞれ24Kバイト，12Kバイトになります．I/Oポートには，アドレス出力用に74LS374が，データ入力用に74LS244がそれぞれ使用されています．また，漢字ROMPAC(PA-7246)には，23256が3個実装されており，JIS第1水準の漢字と非漢字を合わせて3666文字のキャラクタ・パターンが入っています．

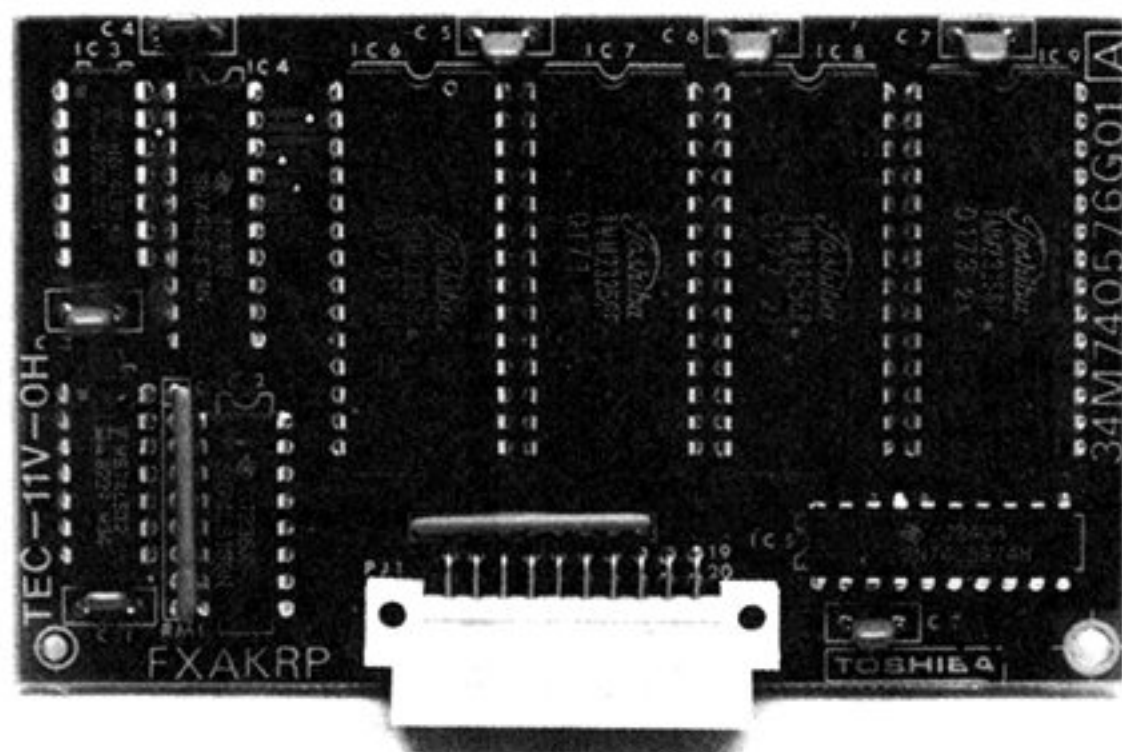



写真1-17
漢字ROMPAC

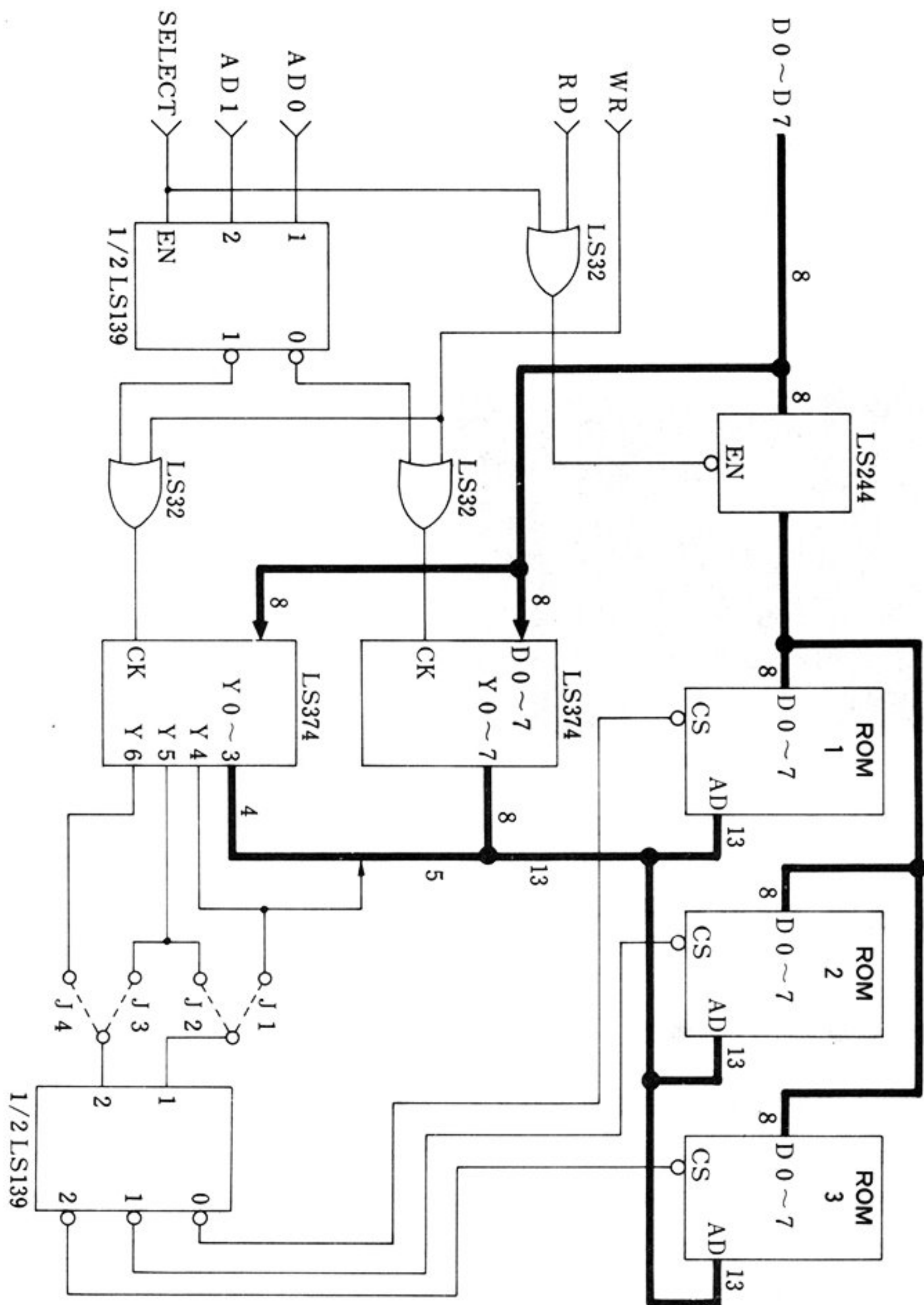
次ページ図1-7-3にROMPAC-2のブロック図を示します．

1-8　拡張ユニット(PA-7300)

1-8-1　拡張ユニットの構成

拡張ユニット(PA-7300)は，RS-232Cインタフェイス，IEEE-488インタフェイスなどの各種ユニバーサル基板を接続することができます．拡張ユニットは6個のユニバーサルカードを格納することができますが，そのうちの1つには拡張ユニットコントローラを入れるのでオプションカードは5枚までということになります．図1-8-1に拡張ユニットのブロック図を，図1-8-2に拡張ユニットの仕様を示します．

1-8-2　拡張ユニットコントローラ

拡張ユニットには6個のカードスロットがありますが，そのうちの1つには拡張ユニットコントローラの基板が入っています．この拡張ユニットコントローラは，各種オプションカードからの割込処理と，ミニディスクユニット・インタフェイスを内蔵しています．ミニディスクユニットは，普通パソピア本体に直接取り付けますが，拡張ユニットとパソピア本体も外部インタフェイスによって接続されます．そのため，ミニディスクユニットは拡張ユニットの方に接続することになります．

ジャンパ線接続表

ROM	2764・2364	2732・2332
J 1・J 3	×	○
J 2・J 4	○	×

図1－7－3　ROMPAC 2の接続図

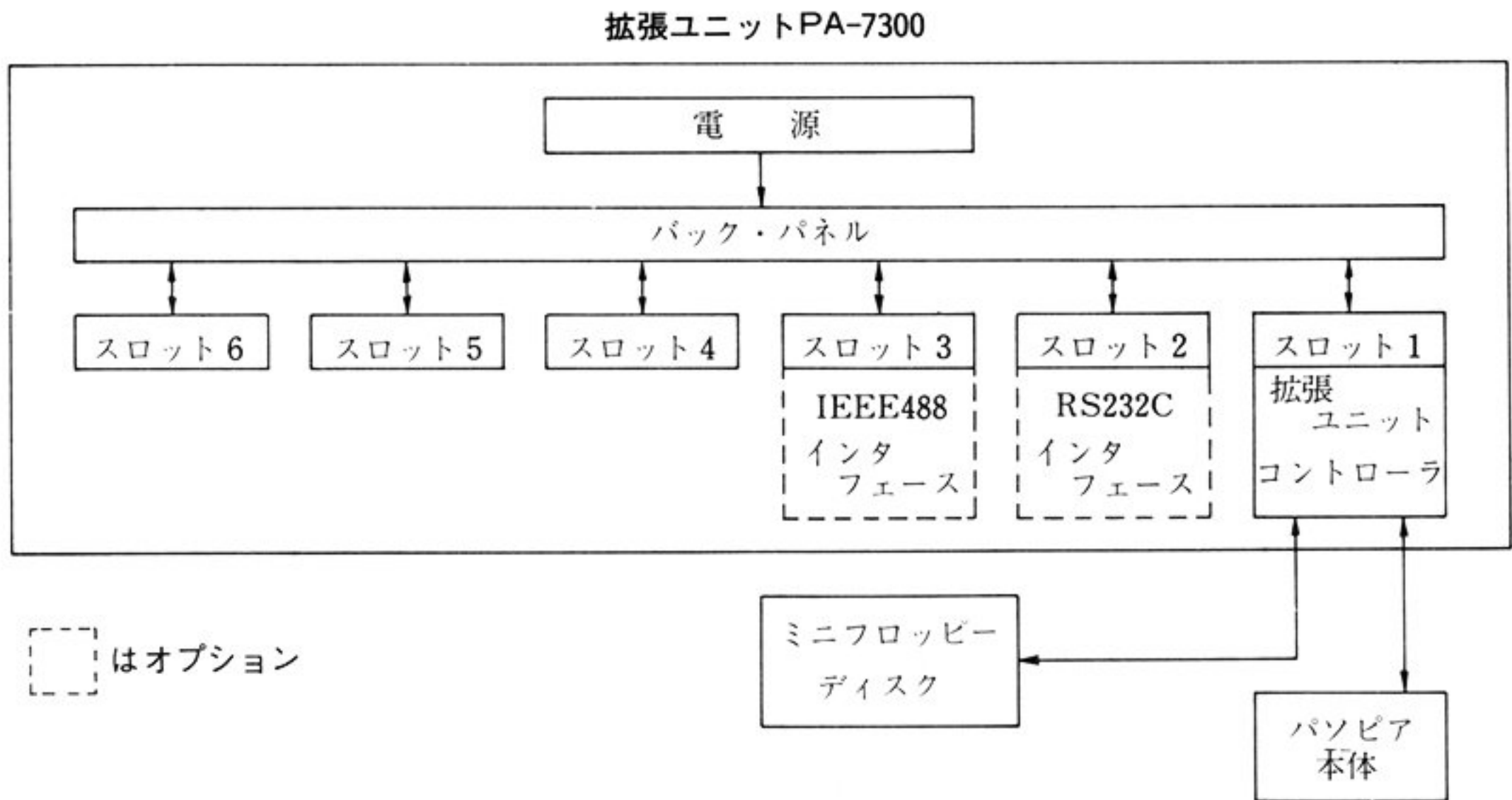



図1－8－1　拡張ユニットのブロック図

外形寸法	420(W)×325(D)×170(H)mm
重　量	約8kg
電　源	AC100V±10%　50／60Hz
消費電力	15W～80W(MAX)
動作温度及び湿度	0℃～35℃, 20%～80%
オプションスロット数	5スロット

図表1―8―2　拡張ユニットの仕様

オプションカードからの割込要求は，拡張ユニットコントローラ上にあるマスク・インヒビットスイッチによってソフトウェアによる割込マスクを可能にするか不可能にするかきめられます．このスイッチがONになっているとオプションカードからの割込は直接本体に伝わります．OFFになっている場合はI/Oポートの4EHのビット7が1のときだけ割込可となり，0のときはオプションカードからの割込は無視されます．図1-8-3に割込マスク制御についての表を示します．

マスクインヒビットスイッチ	I／Oポート4EHのビット7	オプションカードからの割り込み
OFF	0	無
OFF	1	無
ON	0	無
ON	1	有

図表1―8―3　オプションカードからの割り込みスイッチ

割込がイネーブルの状態でオプションカードから割込が発生すると，割込ベクタがFFHの割込がCPUに伝わります．割込の要求源は，ソフトウェアによってI/Oポートの4DHを読むことで，それぞれの処理に分岐して実行されます．また，複数のオプションカードから同時に割込要求があった場合は，拡張ユニットコントローラ基板にもっとも近い物が優先されます．図1-8-4に割込要求識別レジスタの内容を，図1-8-5，に拡張ユニットコントローラのブロック図を，図1-8-6に各コネクタの接続図をそれぞれ示します．

割り込み識別レジスタ（4DH）ビット 7	6	5	4	3	2	1	0	割り込み発生源
0	0	0	0	0	0	0	0	未使用
上位4ビットは必ず“0”				0	0	0	1	IEEE 488インターフェース
				0	0	1	0	RS232Cインターフェース#1
				0	0	1	1	RS232Cインターフェース#2
				0	1	0	0	各種ユニバーサル・ボード
				0	1	0	1	
				0	1	1	0	
				0	1	1	1	
				1	0	0	0	
				～				
				1	1	1	1	

図1―8―4　割り込み要求識別レジスタの内容

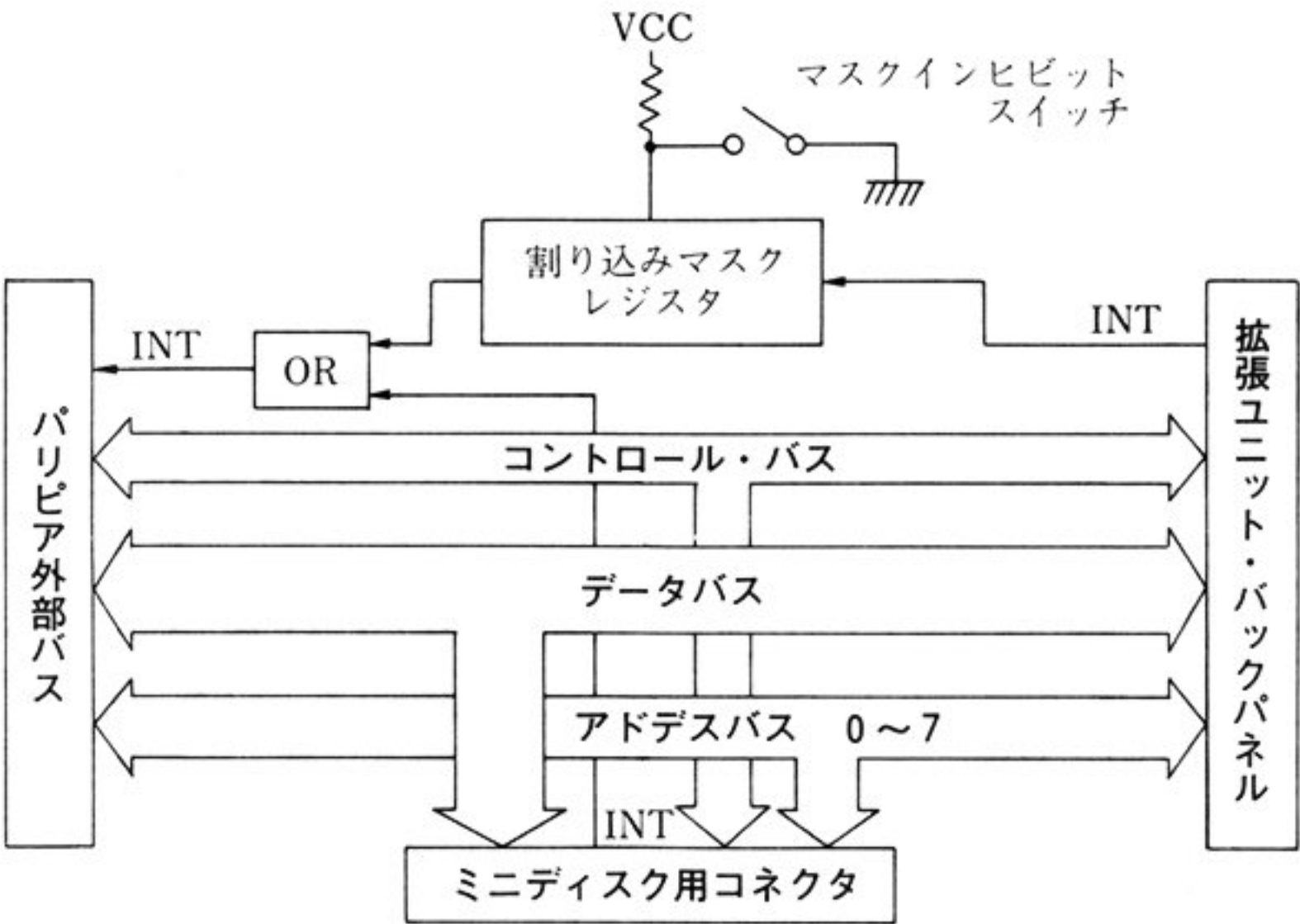



図1―8―5　拡張ユニットコントローラのブロック図

図1－8－6

ピン番号	信号名	I／O	ピン番号	信号名	I／O	ピン番号	信号名	I／O
A01	CN+181		B01	$\overline{\text{8 MHZ}}$	I	C01	CN−181	
A02	GND		B02	$\overline{\text{MI}}$	I	C02		
A03	CN+182		B03	$\overline{\text{IORQ}}$	I	C03	CN−182	
A04	GND		B04	$\overline{\text{RST}}$	I	C04		
A05	CN+183		B05	$\overline{\text{RD}}$	I	C05	CN−183	
A06	GND		B06	$\overline{\text{WR}}$	I	C06		
A07	CN+184		B07	DBO	I／O	C07	CN−184	
A08	GND		B08	DB1	I／O	C08		
A09	CN+185		B09	DB2	I／O	C09	CN−185	
A10	GND		B10	DB3	I／O	C10		
A11	CN+186		B11	DB4	I／O	C11	CN−186	
A12	GND		B12	DB5	I／O	C12		
A13	CN+187		B13	DB6	I／O	C13	CN−187	
A14	GND		B14	DB7	I／O	C14		
A15	DNU		B15	+12V		C15	+12V	
A16	+SU		B16	+5V		C16	+5V	
A17	DNU		B17			C17		
A18	DNU		B18	−12V		C18	−12V	
A19	CN+188		B19	AD88	I	C19	CN−108	
A20	GND		B20	AD81	I	C20		
A21	CN+189		B21	AD82	I	C21	CN−109	
A22	GND		B22	AD83	I	C22		
A23	CN+116		B23	AD84	I	C23	CN−116	
A24	GND		B24	AD85	I	C24		
A25	CN+111		B25	AD86	I	C25	CN−111	
A26	GND		B26	AD87	I	C26		
A27	CN+112		B27	$\overline{\text{B2 NMI}}$	O	C27	CN−112	
A28	GND		B28	$\overline{\text{B1 RQ}}$	O	C28		
A29	CN+113		B29	$\overline{\text{BRDIID}}$	I	C29	CN−113	
A30	GND		B30	$\overline{\text{BIACKI}}$	I	C30		
A31	CN+114		B31	$\overline{\text{BIACKO}}$	O	C31	CN−114	
A32	GND		B32			C32		

拡張ユニット側

注）CN＋1nn →1つ大きいスロット番号のスロットのCnnピンと接続されている．
　　CN－1nn →1つ小さい　　　　　　　　Ann　　〃　　　．

バックパネルのスロット図

PIN	信号名	I／O	PIN	信号名	I／O
01	GND		02	GND	
03	$\overline{SC8MHZ}$	O	04	GND	
05	GND		06	$\overline{MI}$	O
07			08	$\overline{IORQ}$	O
09			10	$\overline{RESET}$	O
11			12	$\overline{RD}$	O
13			14	$\overline{WR}$	O
15			16	$\overline{INT}$	I
17			18	$\overline{NMI}$	I
19			20	DB0	I／O
21			22	DB1	I／O
23			24	DB2	I／O
25			26	DB3	I／O
27			28	DB4	I／O
29			30	DB5	I／O
31			32	DB6	I／O
33			34	DB7	I／O
35			36	AD0	O
37			38	AD1	O
39			40	AD2	O
41			42	AD3	O
43			44	AD4	O
45			46	AD5	O
47			48	AD6	O
49	GND		50	AD7	O

パソピア側

拡張ユニット外部バスコネクタ図

1-9　RS-232Cインタフェイス

1-9-1　概要

パソピア本体には，転送速度が600bpsまでのRS-232Cインタフェイスが内蔵されています(写真1-13)．また，拡張RS-232Cインタフェイスを接続すれば，転送速度を4800bpsまでスピードアップすることができます．

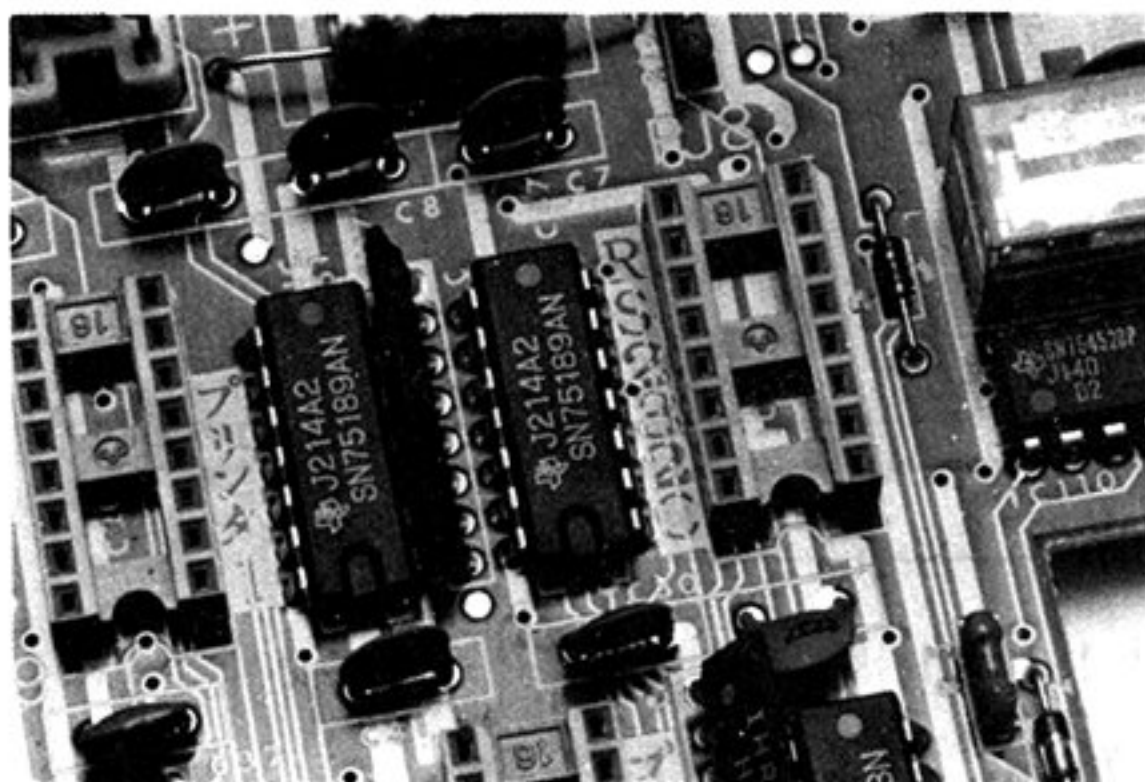



写真 I -18
RS-232Cインタフェイス

本体内のRS-232Cインタフェィスは，すべてソフトウェアによって制御され，ハードウェア的には，パラレルI/Oポートとドライバだけで構成されています．図1-9-1に，RS-232Cインタフェィスのブロック図を示します．

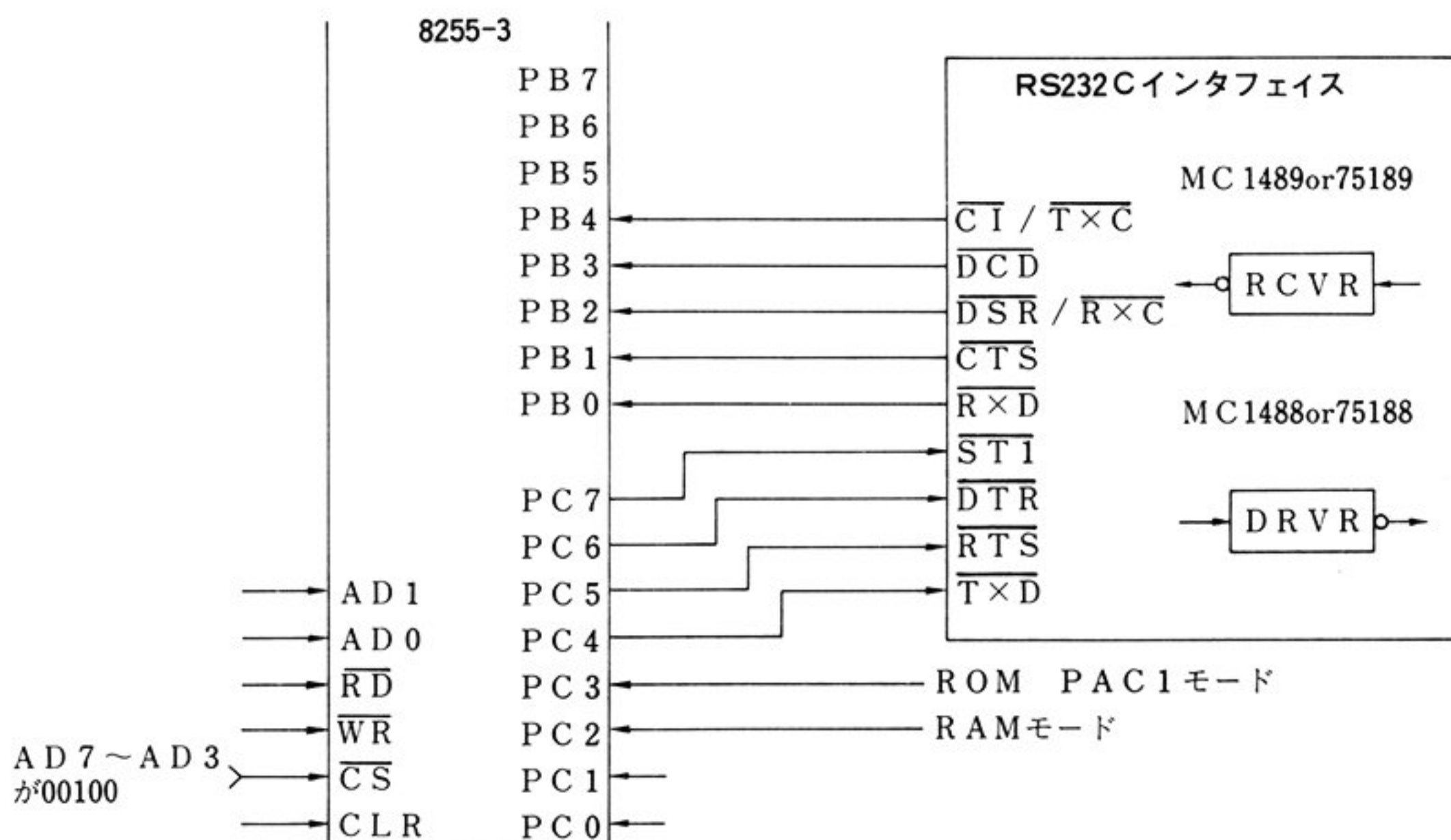



図1－9－1　RS—232Cインタフェィスのブロック図

1-9-2　データ転送のタイミング

データ転送のタイミングは，CTCのタイマ割込によってとられています．データの送信は，図1-9-2のように5回目の割込で次のデータを出力します．一方，データの受信は割込時にフェーズをカウントアップしていき，フェーズの値によってタイミングをとっています．図1-9-3のようにフェーズの値は最初0で，フェーズの0，1，2でスタートビットとし，データビットはフェーズ6でサンプリングします．次の信号がデータのときはフェーズを3にもどし，データが終わるとそのままフェーズがカウントアップされ，フェーズ10でパリティをサンプリングし，

フェーズ14でストップビットとします．その後，データは受信バッファに送られ，フェーズを0にもどし，同じことがくり返えされます．このようにして，RS-232Cの送・受信のソフトウェアによる制御を行っています．

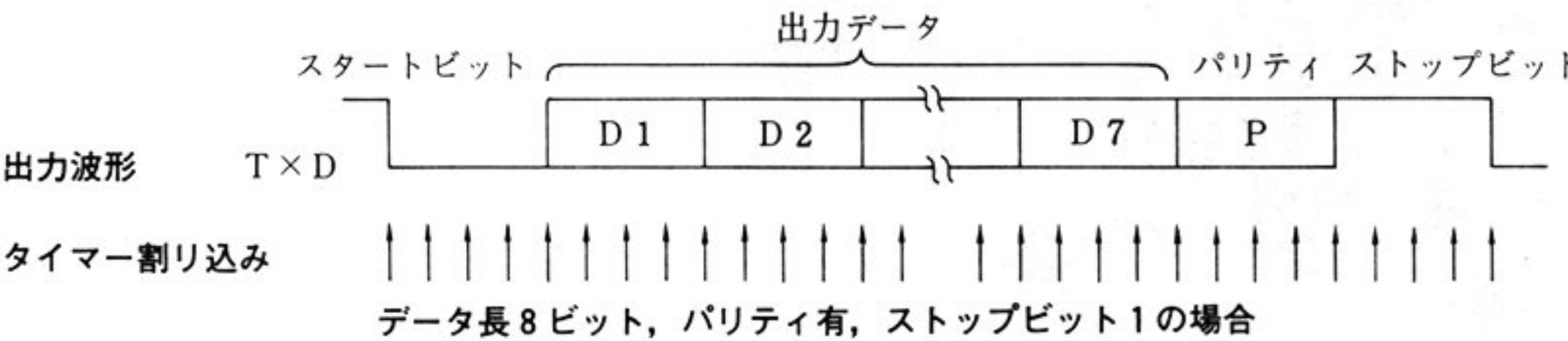



図1—9—2　データ出力のタイミング

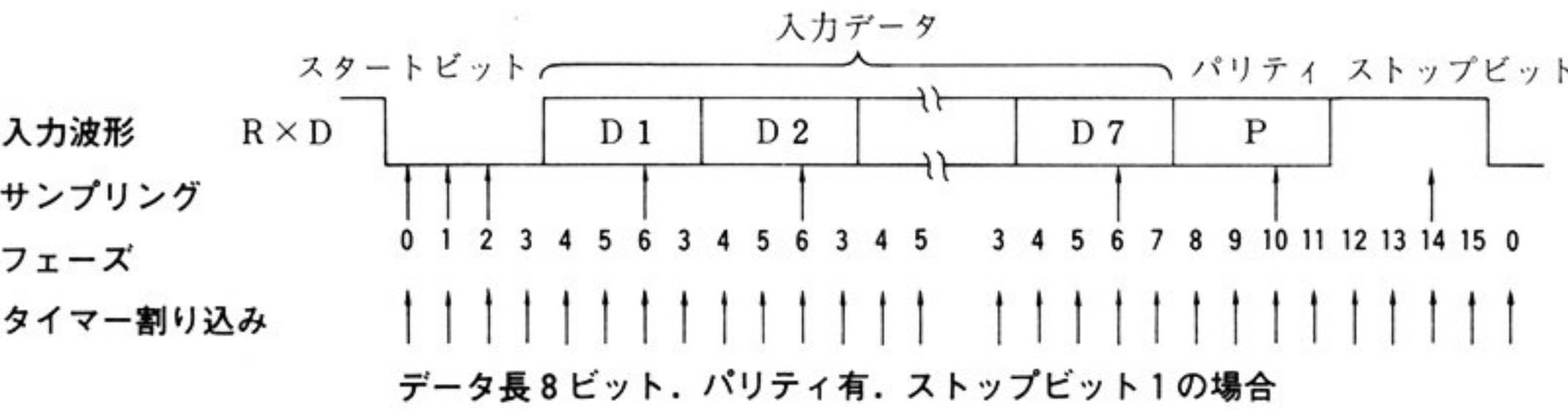



図1—9—3　データ入力のタイミング

1-9-3　回線の接続(ハンドシェイク)

RS-232C回線で他の機器と通信するには，数々の方法がありますが，ここでは最も簡単なコンピュータ間を直接コードでつなぐ方法について説明します．カセットインタフェイスのように，ただデータを送ったり，読みとったりするだけの場合は，各コネクタのGNDをつなぎ，TXDとRXDをそれぞれ接続すれば，一応通信ができます(図1-9-4参照)．

RS-232Cインタフェイスには各種のコントロール信号が入出力されています．これらの信号は，現在送信可であるかとか，回線が接続されているか否かをやりとりするためのもので，これらを接続(ハンドシェイク)することにより，より確実に通信を行うことができます(図1-9-5参照)．

パソピア本体内のRS-232Cコネクタ，およびRS-232Cと規格コネクタの接続図を図1-9-6に，各信号の機能を図1-9-7に示します．

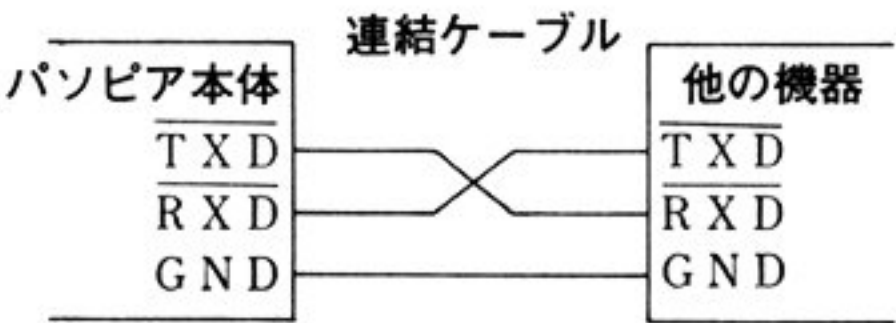



図1—9—4　最も簡単な接続法

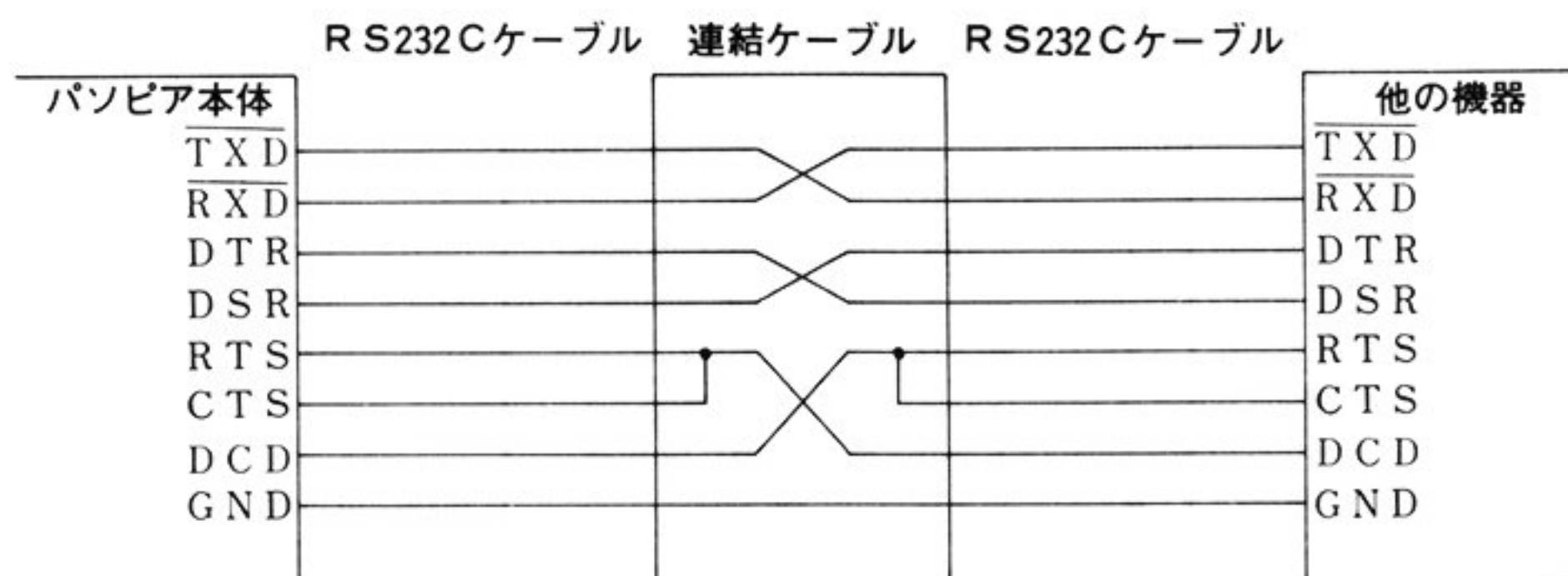
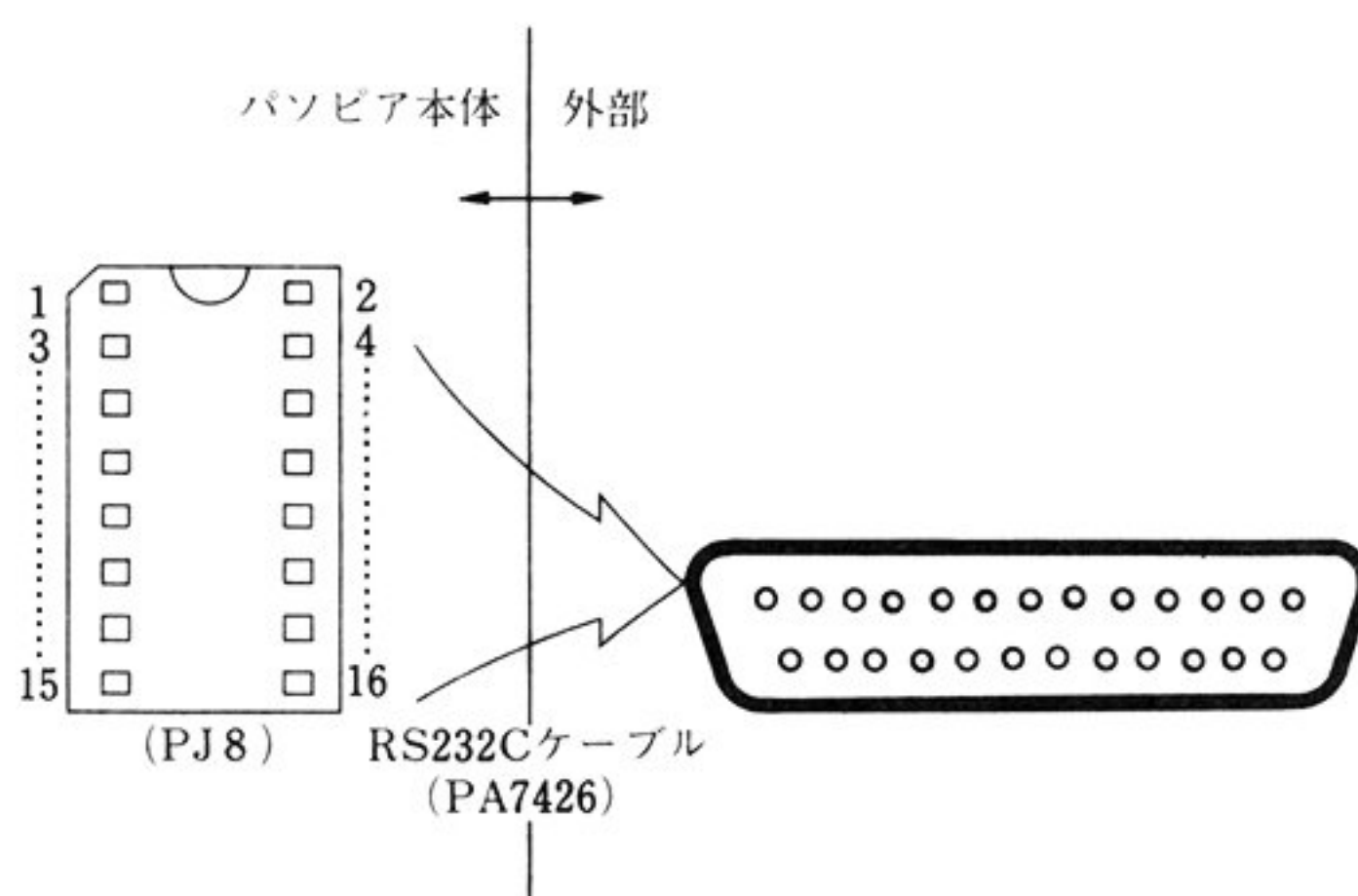



図１—９—５　ＲＳ-232Ｃ回線の接続

●RS232C

本体内のICソケット

ピン番号	信号名	I／O	ピン番号	信号名	I／O
1	FGNC		2	$\overline{\text{TXD}}$	O
3	$\overline{\text{RXD}}$	I	4	RTS	O
5	CTS	I	6	DSR	I
7	SCCND		8	DCD	I
9			10	RXC	I
11			12	DTR	O
13			14	CI	I
15	TXC	I	16	STI	O

パソピア側

図１—９—６　RS232Cコネクタ図

25ピン232C用コネクタ

ピン	信号名	ピン	信号名
1	FG(NC)	14	
2	$\overline{\text{TXD}}$	15	$\overline{\text{TXC}}$
3	$\overline{\text{RXD}}$	16	
4	RTS	17	RXC
5	CTS	18	
6	DSR	19	
7	SG(LND)	20	DTR
8	DCD	21	
9		22	CI
10		23	
11		24	STI
12		25	
13		26	

外部機器側

図1－9－7　各信号の機能

ピン	信号名	機能
1	FG	(NC)フレームGND
2	TXD	送信データ
3	RXD	受信データ
4	RTS	送信要求
5	CTS	送信可
6	DSR	データ・セットレディ
7	SG	GND
8	DCD	データチャンネル受信キャリア検出
15	TXC	送信信号エレメント・タイミング(DCE)
17	RXC	受信信号エレメント・タイミング
20	DTR	データ・ターミナル・レディ
22	CI	被呼表示
24	STI	送信信号エレメント・タイミング(DTE)

第2章

T-BASICの内部構造

第２章　T-BASICの内部構造

2-1　メモリ内部の状態

この章では，パソピアに使用されている．T-ROMBASIC(Ver1.1)を中心に解説します．

このT-BASICは，Microsoft-BASIC(Ver5.0)にグラフィック機能や，RAMPAC処理機能等を追加させたもので，特徴としては，変数名を40字まで判断する，WHILE・WEND命令を持つ，などが上げられます．

また，第１章で述べたように，V-RAMのアクセスにDMAを行わないため，他のMicrosoft系BASICに比べ，高速(1.5倍程度)になっています．(月刊アスキー　57年10月号参照)

2-1-1　メモリ・マップ

T-BASICでは，ROM版Ver1.0，Ver1.1，ともに，ROM32Kバイト，RAM32Kバイトを使用しています．そのため，V-RAMを除いた64KバイトのRAMの半分にあたる32Kバイトは未使用です．

これに対し，DISK版では，ROM版で未使用であったRAMにDISK-BASICが書き込まれ，64KRAMシステムとして使用されています．(1-1-3参照)

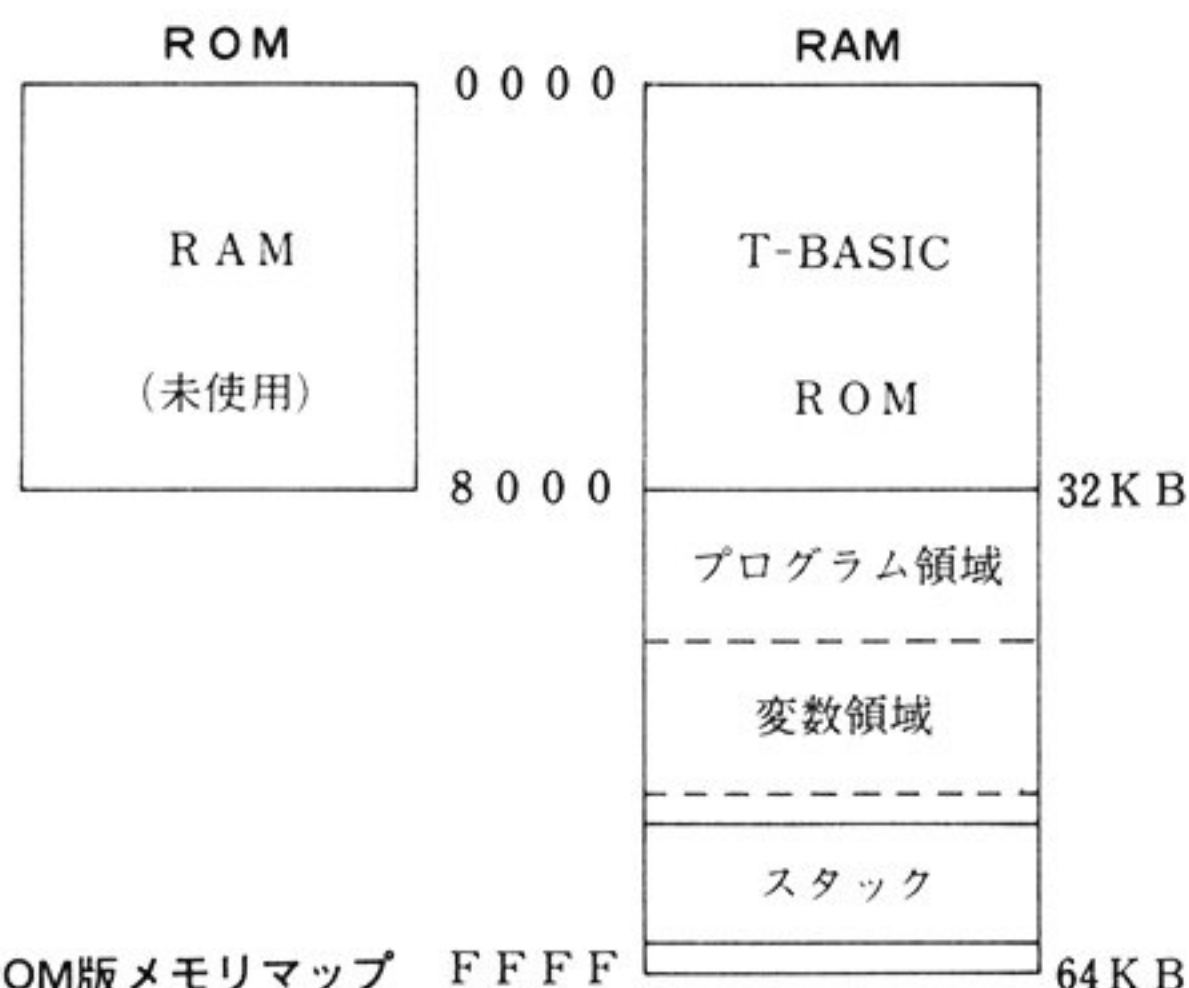



図2－1－1(a)　ROM版メモリマップ

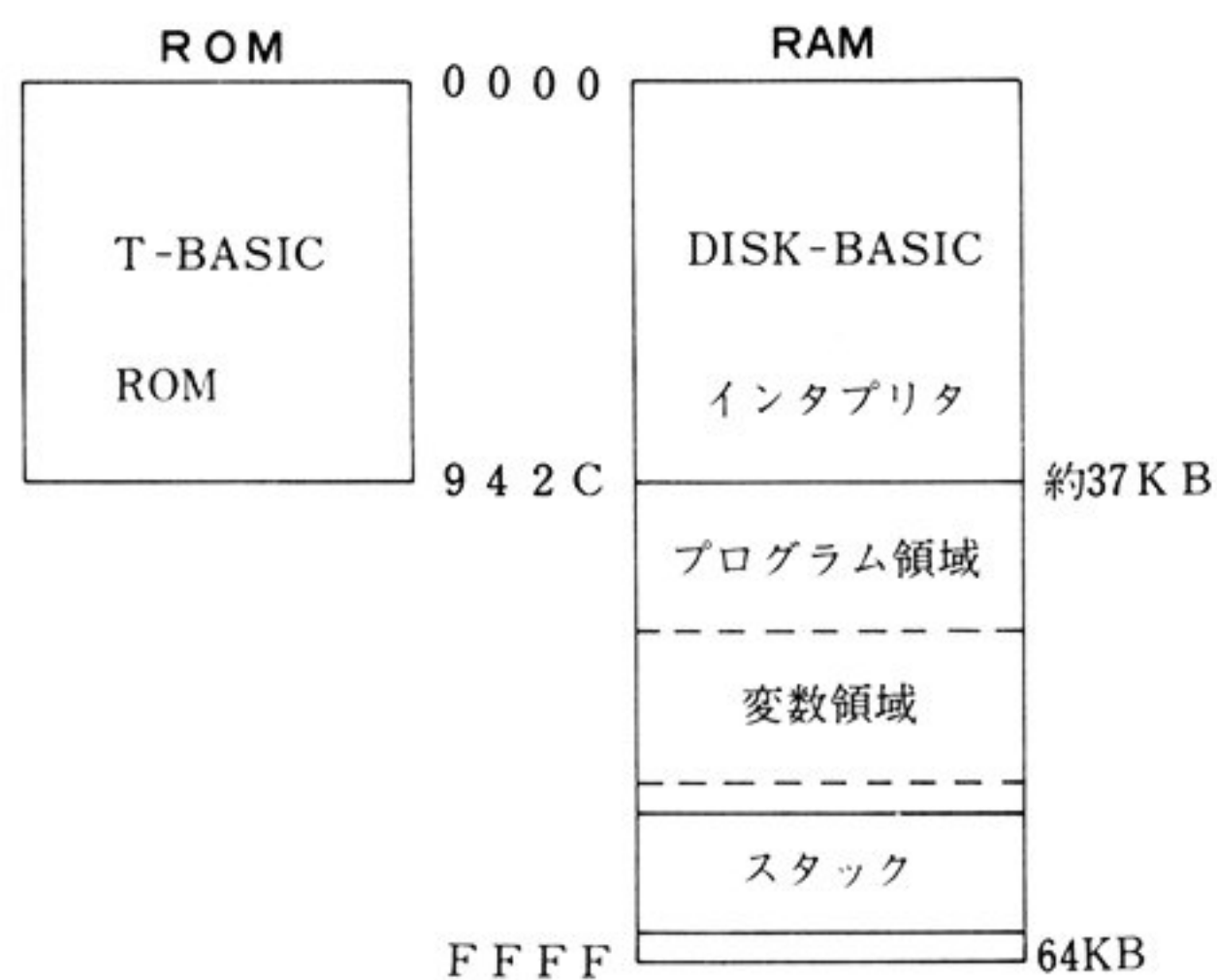



図2—1—1（b）　DISK版システムメモリマップ

そのうち，8000(DISK版は942C)～FFFFHのメモリ・マップを示します．

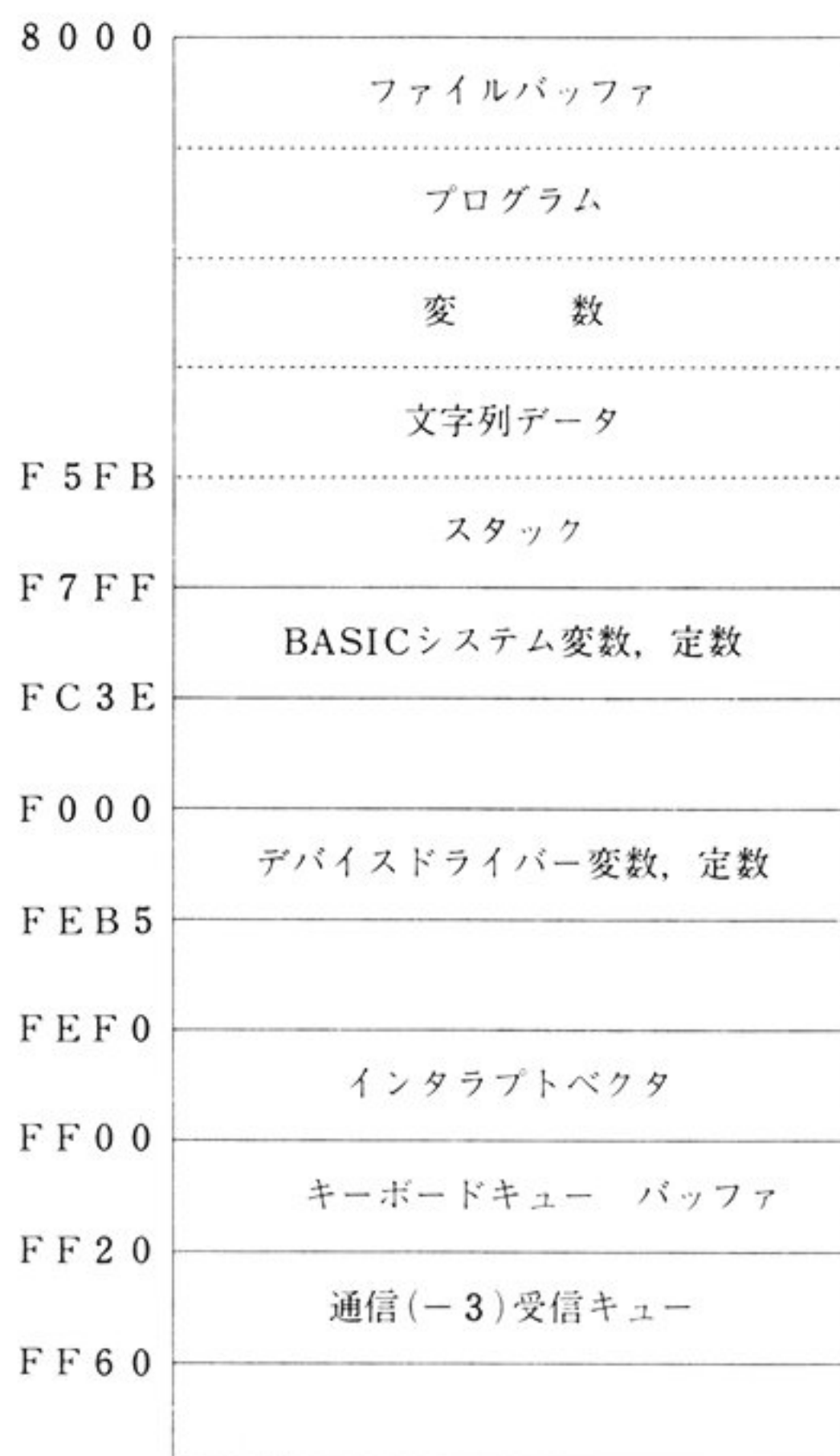
図2—1—2(a)　ROM版ユーザ・エリア

アドレス	内容
	通信(−3)モード
FFEB	通信(−3)速度
FFEC	プリンタ監視時間
FFED	システムサブルーチン・ジャンプテーブル
FFFF	

図2−1−2(b)　DISK版ユーザ・エリア

2-1-2 プログラム領域

T-BASIC(DISK)では，ユーザ・エリアは，8000H(DISK版は942CH)からF5FBH(DISK版はFCE3H)までとなっています．このユーザ・エリアは次の図のように割り当られています．

図2－1－3(a) ROM版ユーザ・エリア

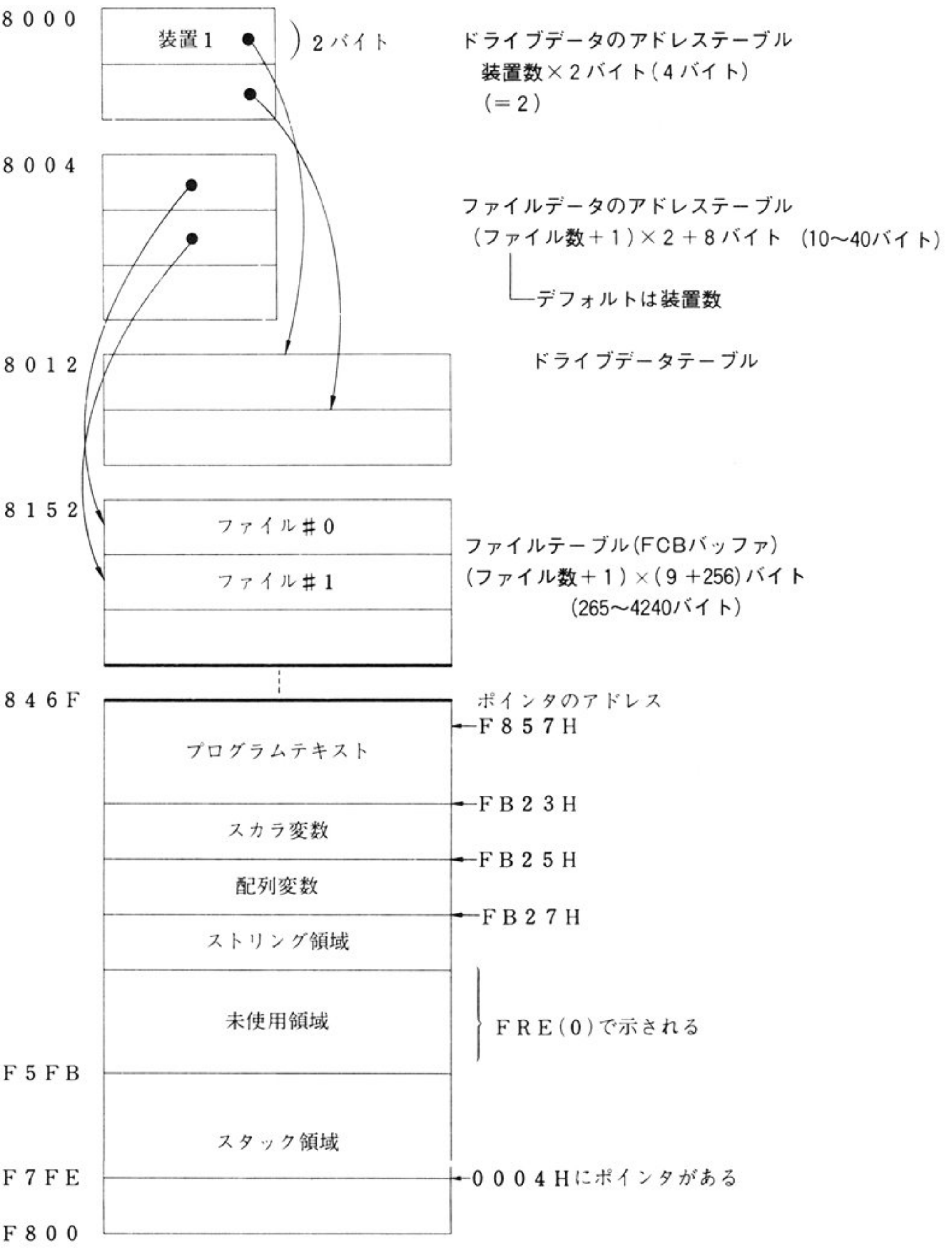



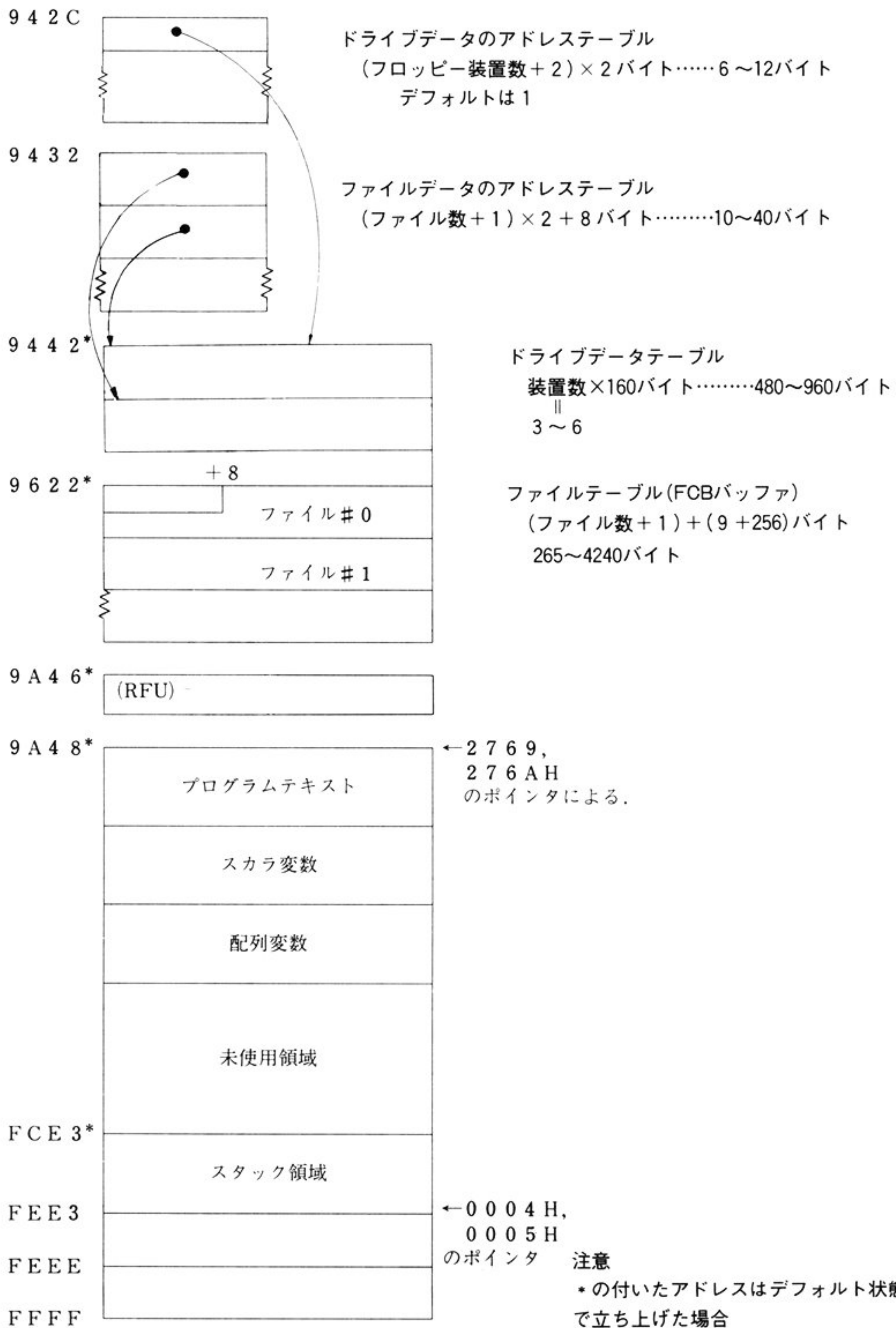
図2－1－3(b) DISK版ユーザ・エリア

このうちのプログラム領域の大きさは，次の図表のように，電源投入時もしくはリセット時に入力するファイル数によって決まります．

図2－1－4　プログラム領域の大きさ(ROM版)

ファイル数	プログラム領域サイズ	テキスト先頭アドレス
0	29,600バイト	8259H
1	29,333	8364
2	29,066 (デフォルト)	846F (デフォルト)
3	28,799	852A
4	28,532	8685
5	28,265	8790
6	27,998	8893
7	27,391	89A6
8	27,464	8AB1
9	27,197	8BBC
10	26,930	8CC7
11	26,663	8DD2
12	26,396	8EDD
13	26,129	8FE8
14	25,862	90F3
15	25,594	91FE

この表の中の〝デフォルト〟とは，〝How many　files(0～15)?〟に対して，リターンキーのみを押した場合の値です．

スタック領域は，F7FA～F5FBHの512バイトとなっています．プログラム領域は次の式で計算

できます.

ROM版

プログラム領域サイズ＝29600-267×F(バイト)　　　F：ファイル数

DISK版

プログラム領域サイズ＝26212-267×F－162×D(バイト)　　　F：ファイル数
D：ドライブ数

プログラムの格納されるアドレス等は，ポインタによって指定されています．ポインタは，それぞれ次のアドレスにあります．

図2－1－5　T-BASIC／T-DISK　BASICポインター覧

内部シンボル	内　容	アドレス	
		ROM版	DISK版
INITSA	BASICで使うRAMの最下位アドレス	＝8000	＝942C
DRVTAB	ドライブテーブル(アドレス)の先頭アドレス	FC84	5A68
FILTAB	ファイルテーブル(アドレス)の先頭アドレス	FC82	5A66
TXTTAB	プログラムテキストの先頭アドレス	F857	2769
VARTAB	スカラ変数の先頭アドレス	EB23	2A36
ARYTAB	アレイ変数の先頭アドレス	FB25	2A38
STREND	ストリング領域の先頭アドレス	FB27	2A3A
FRETOP	ストリング領域の未使用先頭アドレス	FAFC	2A0E
MEMSIZ	ストリング領域の最終アドレス	FAD7	29E9
TOPMEM	BASICスタックの最終アドレス(RAM上限)	F853	2765
MAXMEM	機械語プログラム領域の上限アドレス	0004	0004
DATPTR	DATA文ポインタ	FB29	2A3C

このポインタの内容は，PEEK関数により調べることができます．ROM版で，ポインタを調べてみましょう．電源投入時に，ファイル数を０にして，ポインタをPEEKで読みます．

プログラムを入力する前は，次のようになります．

```
How many files(0-15)? 0
Toshiba T-BASIC Ver 1.0
(c) 1982 by Microsoft
29600 Bytes free
Ok
?hex$(peek(&Hf857)),hex$(peek(&Hf858
))
59            82
Ok
?hex$(peek(&Hfb23)),hex$(peek(&Hfb24
))
5B            82
Ok
?hex$(peek(&Hfb25)),hex$(peek(&Hfb26
))
5B            82
Ok
?hex$(peek(&Hfb27)),hex$(peek(&Hfb28
))
5B            82
Ok
```

ポインタは，（下位１バイト），（上位１バイト）の順番で書かれています．プログラムを打ち込んで変化を見てみましょう．

```
10 REM  sample 1
20 DIM A(10)
30 A=1:B=2
40 FOR I=1 TO 10
50   A(I)=A*B+1
60 NEXT I
70 A$="end"
80 PRINT A$
90 END
Ok
?hex$(peek(&Hf857)),hex$(peek(&Hf858))
59            82
Ok
?hex$(peek(&hfb23)),hex$(peek(&Hfb24))
C7            82
Ok
?hex$(peek(&Hfb25)),hex$(peek(&Hfb26))
C7            82
Ok
?hex$(peek(&Hfb27)),hex$(peek(&Hfb28))
C7            82
Ok
```

プログラム開始のポインタを除いたポインタが変化しました．このプログラムを実行すると，さらに変化するポインタがあります．

```
RUN
end
Ok
?hex$(peek(&Hf857)),hex$(peek(&Hf858))
59            82
Ok
?hex$(peek(&hfb23)),hex$(peek(&Hfb24))
C7            82
Ok
?hex$(peek(&Hfb25)),hex$(peek(&Hfb26))
E6            82
Ok
?hex$(peek(&Hfb27)),hex$(peek(&Hfb28))
1B            83
Ok
```

プログラムを実行すると，配列領域の始まりのポインタとフリーエリアの始まりを示すポインタが変化しました．

各ポインタの変化を次の図表に示します．

図2—1—6　各ポインタの変化(ROM版)

	アドレス	電源投入時	プログラム 実行前	プログラム 実行後
プログラムの開始	F857. 58	8257		
変数領域の始まり	FB23. 24	825B	82A9	
配列変数領域の始まり	FB25. 26	825B	82A9	82B8
フリーエリアの始まり	FB27. 28	825B	82A9	83F0

2-1-3　中間言語(トークン)

BASICの各命令(キーワード)は，メモリの効率を上げ，実行速度を向上させるために，メモリ上では，1バイトもしくは2バイトの中間言語(トークン)の形で書き込まれています．

次にキーワードと中間言語の対応表を載せます．この対応表はROM内にあり，T-ROMBASIC Ver 1.0では1995Hから，Ver1.1では19ECHから，T-DISKBASICでは20A0Hから置かれています．

図2—1—7　キーワードと中間言語の対応表

KEYWORD	TOKEN	KEYWORD	TOKEN	KEYWORD	TOKEN
AUTO	AA	FPOS	FF A8	PRESET	CE
AND	F4	GOTO	89	POINT	FF D2
ABS	FF 86	GO TO	89	PAINT	CC
ATN	FF 8E	GOSUB	8D	PLAY	D6
ASC	FF 95	GET	BC	RETURN	8E
ATTR$	E8	HEX$	FF 9A	READ	87
BSAVE	D2	INPUT	85	RUN	8A
BLOAD	D1	IF	8B	RESTORE	8C
CLOSE	BF	INSTR	E4	REM	8F
CONT	99	INT	FF 85	RESUME	A8
CLEAR	92	INP	FF 90	RSET	C6
CLOAD	9B	IMP	F8	RIGHT$	FF 82
CSAVE	9A	INIT	D4	RND	FF 88
CSRLIN	E7	INKEY$	EB	RENUM	AB
CINT	FF 9D	KEY	FF D3	RANDOMIZE	B8
CSNG	FF 9E	KILL	C4	SCREEN	FF D0
CDBL	FF 9F	KANJI	FF D5	STOP	90
CVI	FF A1	LPRINT	9D	SWAP	A4
CVS	FF A2	LLIST	9E	SET	BE
CVD	FF A3	LPOS	FF 9B	SAVE	C7
COS	FF 8C	LET	88	SPC(	DE
CHR$	FF 96	LOCATE	C9	STEP	DB
CALL	B3	LINE	B0	SGN	FF 84
COMMON	B5	LOAD	C0	SQR	FF 87
CHAIN	B6	LSET	C5	SIN	FF 89

COM	FF D4	LIST	93	STR$	FF 93
CIRCLE	CA	LFILES	C8	STRING$	E2
COLOR	CF	LOG	FF 8A	SPACE$	FF 98
CLS	9F	LOC	FF A6	SOUND	D0
DELETE	A9	LEN	FF 92	THEN	D9
DATA	84	LEFT$	FF 81	TRON	A2
DIM	86	LOF	FF A7	TROFF	A3
DEFSTR	AC	MOTOR	CB	TAB(	DA
DEFINT	AD	MERGE	C1	TO	D8
DEFSNG	AE	MOD	F9	TAN	FF 8D
DEFDBL	AF	MKI$	FF A9	TERM	D3
DSKO$	B9	MKS$	FF AA	TIME	FF D1
DEF	97	MKD$	FF AB	USING	E3
DSKI$	E9	MID$	FF 83	USR	DC
DSKF	FF A4	NEXT	83	VAL	FF 94
DRAW	D5	NAME	C3	VARPTR	E6
ELSE	A1	NEW	94	WIDTH	A0
END	81	NOT	DF	WAIT	96
ERASE	A5	OPEN	BA	WHILE	B1
EDIT	A6	OUT	9C	WEND	B2
ERROR	A7	ON	95	WRITE	B4
ERL	E0	OR	F5	OR	F6
ERR	E1	OCT$	FF 99	+	EF
EXP	FF 8B	OPTION	B7	-	F0
EOF	FF A5	OFF	EA	*	F1
EQV	F7	PRINT	91	/	F2
FOR	82	PUT	BD	^	F3
FIELD	BB	POKE	98	¥	FA
FILES	C2	POS	FF 91	'	E5
FN	DD	PEEK	FF 97	>	EC
FRE	FF 8F	PORT	FF 9C	=	ED
FIX	FF A0	PSET	CD	<	EE

2-1-4　中間言語の処理

BASICインタプリタがプログラムを実行するときには，中間言語を解釈し，ジャンプテーブルに従って処理ルーチンをコールして実行しています．ジャンプテーブルはROM版Ver1.0では184AH，Ver1.1では18A2Hから中間言語の小さいものから順に書き込まれています．

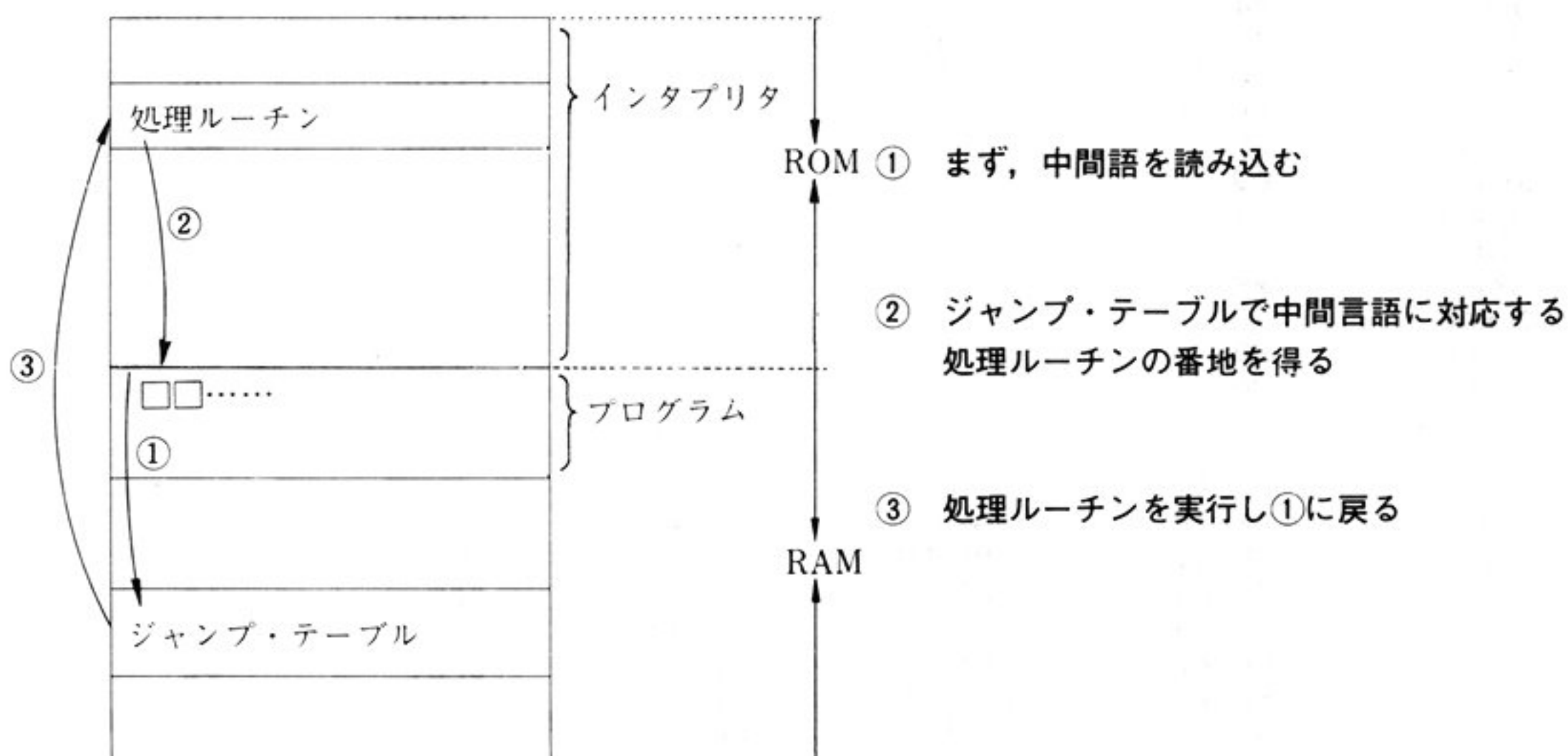



図2―1―8　中間言語の処理

なお処理ルーチンの一覧表はAPPENDIXを参照して下さい．

2-1-5　プログラムの格納状態

T-BASICの各命令はメモリ上に16進数2桁に圧縮されて書き込まれています．その状態を調べましょう．

T-BASICに機械語モニタがないのでその代用としてAPPENDIXにモニタ機能プログラムを載せましたのでこれを利用します．

また，PASOPIA用のデバッガとして，DD-PS(ダイナミック・デバッガ)がアスキー・コンシューマ・プロダクツから発売されていますので，内部を詳しく調べたい方は利用されるとよいでしょう．

付属のプログラムの行番号は10000から始まっているので，行番行が10000より小さいBASICのプログラムと共存できます．

このプログラムを使って先ほどのサンプルプログラムがメモリに書き込まれている様子を調べてみましょう．

```
8250 : 00 00 00 00 00 00 00 00  00 69 82 0A 00 8F 20 20   ........ .i▬..+.. 55
8260 : 00 00 00 00 00 00 00 00  00 75 82 14 00 86 20 41   sample.1 .u▬..■.A A7
8270 : 00 00 00 00 00 00 00 00  00 41 ED 12 3A 42 ED 13   (..).... .AO.:BO. 39
8280 : ED ED ED ED ED ED ED ED  ED 12 20 D8 20 0F 0A 00   ...(...I O..ﾘ.... 57
8290 : 49 49 49 49 49 49 49 49  49 29 ED 41 F1 42 EF 49   ..2...A( I)OAﾑB\I 1B
82A0 : 00 00 00 00 00 00 00 00  00 B6 82 46 00 41 24 ED   ...(...I .ｶ▬F.A$O 45
82B0 : 50 50 50 50 50 50 50 50  50 00 91 20 41 24 00 C5   "end"... P.┬.A$.ﾅ 19
82C0 : 27 27 27 27 27 27 27 27  27 3A 8F E5 FE 78 78 78   .Z...... ':+◣.xxx 58
82D0 : 78 78 78 78 78 78 78 78  78 78 78 78 78 78 78 78   xxxxxxxx xxxxxxxx D2
82E0 : 78 78 78 78 78 78 78 78  78 78 78 00 13 83 1A 27   xxxxxxxx xxx..▬.' 61
```

このように，BASICのプログラムは，2桁の16進数に変換されて，メモリに書き込まれています．以下にその解説を載せます．

図2―1―9　プログラムの格納状態

6 9	次の行の開始アドレスを示すリンクポインタ（次の行は8269Hから）	2 0	スペース
8 2		4 1	"A"
		2 8	"C"
0 A	行番号000AH → 10	0 F	次のバイトは1バイトの整数であることを示す識別コード
0 0			
8 F	REMを示す中間言語	0 A	0AH＝10
2 0	スペース "␣"	2 9	") "
2 0	スペース	0 0	行の終り
7 3	"s"	8 1	リンクポインタ
6 1	"a"	8 2	
6 D	"m"	1 E	行番号30
7 0	"p"	0 0	
6 C	"e"	4 1	"A"
6 5	"l"	E D	"＝" を示す
2 0	"␣"	1 2	1を示す

31	"1"
00	行の終りを示している.
75	リンクポインタ
82	
14	行番号20
00	
86	DIMを示す
3A	":"
42	"B"
ED	"="
13	2を示す
00	行の終り

2バイトで表されるリンクポインタなどの上位と下位の順序が，逆になっている点に注意して下さい．

また，T-BASICでは，GOTO・GOSUB・THENに続く行番号は，実行時に飛び先を調べて書き換えています．

先ほどのプログラムに，

```
5 GOTO 10000
```

を加えて調べてみましょう．

まず，実行前の状態を見るために，RUN 10000として，8259～826F番地を見て下さい．次に実行後の状態を見るために，RUNとして比べてみます．

```
8250 : 00 00 00 00 00 00 00 C6  00 63 82 05 00 89 20 0E   ........ .c▬..|.. 39
8260 : 10 27 00 73 82 0A 00 8F  20 20 73 61 6D 70 6C 65   .'.s.... ..sample 69

8250 : 00 00 00 00 00 00 00 C6  00 63 82 05 00 89 20 0D   ........ .c▬..|.. 38
8260 : CE 82 00 73 82 0A 00 8F  20 20 73 61 6D 70 6C 65   ...s.... ..sample 82
```

5行のGOTO文を実行する前と後で，825FHからの値が，0E　10　27から，0D　CE　82，に変っているのがわかります．0Eは，行番号を示す識別コードで，0Dはアドレスを示す識別コードです．

識別コードについては後述しますので，ここでは，飛び先の行番号10000の16進表現2710Hが，飛び先の番地82CEHに変わっている点に注目して下さい．

識別コードとはプログラムエリア中で数値の前につけられていて，それに続く数値が何であるかを示すもので，インタプリタがキーワードや変数名と区別するために使用されます．

識別コードは次のようになっています．

図2—1—10　識別コード

識別コード	意　　味
0B	(&O)以下の2バイトは，プログラム中の8進数を16進数に変換したものです．(−32768〜32767…整数)
0C	(&H)以下の2バイトは，16進数です．(−32768〜32767…整数)
0D	以下の2バイトは，行番号などを，実際のメモリ上の飛び先アドレス(16進)に変換したものです．
0E	以下の2バイトは，GOTO, GOSUB, THENの後に続く，行番号(0〜65535)を16進に変換したものです． 注) 0Eの後の行番号は，一度プログラムを実行すると，実際のアドレスに変換され，識別コードも"0D"に変わります．
0F	以下の1バイトは，10〜255の数値を16進で表わしたものです．(0A〜FF)
11〜1A	識別コードそのものが0〜9の数値を表わします．(11→0，12→1，13→2，……，19→8，1A→9)
1C	以下の2バイトは，整数(−32768〜32767)を16進数に変換したものです．
1D	以下の4バイトは，浮動小数点表記の単精度実数です．
1F	以下の8バイトは，浮動小数点表記の倍精度実数です．

また，プログラム中に書かれた数値は，その型により変換されて格納されます．

整数は，整数型の識別コード1Cに続いて，数値が16進2バイトに変換され，下位，上位の順で格納されます．

単精度数は，識別コード1Dに続いて，数値が，4バイトの浮動小数点表記法により表現され格納されます．

倍精度数は，識別コードが1Fに，数値の表現が8バイトになった以外は，単精度数と同じです．

負符号はどの場合でも，識別コードの前に入ります．

具体例を示しましょう．次のプログラムを打ち込んで，メモリの内部を見てみます．

```
10 A!=123456!
20 B#=123.456789012#
30 C%=12345

LIST -30
10 A!=123456!
20 B#=123.456789012#
30 C%=12345
OK
RUN
[[[  TINY MONITOR          Ver. 1.0          ]]]
[[[              by J.Azuma                  ]]]
[[[               Oct.'82                    ]]]
]D   Dump memory
START ADDRESS? 8250
  END ADDRESS? 828F
8250 : 00 00 00 00 00 00 00 C6  00 66 82 0A 00 41 21 ED   ........ .f...A!. D9
8260 : 1D 00 20 71 91 00 77 82  14 00 42 23 ED 1F AE 6C   ...q..w. ..B#...l B9
8270 : D7 3F E0 E9 76 87 00 82  82 1E 00 43 25 ED 1C 39   .?..v... ...C%..9 9A
8280 : 30 00 A9 82 10 27 3A 8F  E5 FE 78 78 78 78 78 78   0....':. ..xxxxxx 10
]
```

このように，数値の種類によって，格納のされ方が異なるのが，わかります．

ここで扱った浮動小数点表記法については，変数値にも使われていますので，その項で説明したいと思います．

2-1-6 変数の格納状態

プログラムの中で使用されている変数の値も，型により変換されて格納されるのは数値の場合と同じです．

しかし，変数は定数のように数値と異なり，プログラムのテキストエリアとは別の変数エリアに書き込まれています．

変数値の格納されている場所を示すポインタは，図2-1-6に示しましたので，それを使って格納のされ方を見てみましょう．

```
10 '***
20 '*** PROGRAM 2-1-8
30 '***
40 WIDTH 80:CLS
50 DIM AA$(10),BA(10)
60 FOR LOOP=1 TO 10
70 AA$(LOOP)=STRING$(LOOP,32+LOOP)
80 BA(LOOP)=I
90 TOTAL=TOTAL+LOOP
100 NEXT LOOP
110 CA$=STRING$(255,"*")
120 DA%=255
130 PRINT"SET COMPLETE"
```

上のプログラムを打ち込んで，実行したのち，格納状態を見て下さい．

まず，単純変数から調べます．

```
単純変数領域
 D8330,835F
8330 4D 50 4C 45 54 45 22 00 00 00 04 4C 4F 02 CF D0  MPLETE"....LO...
8340 00 00 30 84 04 54 4F 03 D4 C1 CC 00 00 5C 86 03  ..0..TO......¥..
8350 43 41 00 FF CA 9C 02 44 41 00 FF 00 03 41 41 00  CA.....DA....AA.
#P
```

このように，A，B，Cの値が，整数，単精度，倍精度の別に，型変換されているのがわかります．ここで図2-1-11に変数の格納状態についてまとめます．

変数の型	変数名の先頭2バイト	変数名の残りベイト数	変数名の残り	データ部分	合計
整数型	02	2バイト	1バイト	nバイト	2バイト（VARPTR関数はこのアドレスを指す）	6＋nバイト
単精度型	04	2バイト	1バイト	nバイト		8＋nバイト
倍精度型	08	2バイト	1バイト	nバイト	8バイト	12＋nバイト
文字列型	03	2バイト	1バイト	nバイト	文字数 ポインタ	7＋nバイト

ストリング　ディスクリプタ
ポインタは文字列の格納アドレスを指す．

図2－1－11　型による変数の格納状態

ところで，今まで度々出てきた，浮動小数点表記法について説明しましょう．
例として，LOOP＝303!がどう格納されるか，その手順を追ってみます．

04	4C	4F	02	CF	D0	00	80	17	89
単精数	L	O	残り2文字	O（最上位を立ててある）	P	浮動小数点表記部分			

図2－1－12a　LOOP！＝303！の格納状態

LOOPの格納状態は上の通りです．
浮動小数点表記部は次のようになっています．

00	80	17	89
仮数部下位	仮数部中位	仮数部上位	指数部（81Hが加えられている）

図2－1－12b　浮動小数点表記部分

この格納の手順は次のようになります．

図2－1－13　格納の手順

LOOP＝303！

① 数値を2進数に変換する ―→ $303=100101111_2$

② 2を底とする指数表現になおす ―→ $100101111=1.00101111\times2^8$

③ 指数(この場合8)に81Hを加え指数部とする ─→ 8＋81H＝89H

④ 仮数部の小数部を取り出す ─→ 1.00101111→00101111

⑤ 左に正負に対応する値を付け加える(正→0，負→1) ─→ 00101111→000101111

⑥ 24ビット(倍精度は56ビット)になるよう後に0をつける ─→ 000101111000000000000000

⑦ 8ビットごとに16進になおす ─→ 00010111＝17H
10000000＝80H
00000000＝00H

⑧ 下位，中位，上位の順になおし指数部を後ろにおく ─→

00	80	17	89

また，逆に，格納された値から10進の値を知るためには，この手順の逆を行えばよいのですが，そのかわりに，CVS命令を使って，

```
CVS(CHR$(&HW)+CHR$(&HX)+CHR$(&HY)+CHR$(&HZ)
```

を行えばよいでしょう．W, X, Y, Zには16進で浮動小数点表記部の内容を順に入れます．この型の数値の計算はすべて，ワークエリア内のFACと呼ばれる，浮動小数点アキュムレータを通じて行われます．

次は配列変数です．配列変数の開始アドレスは先ほど調べたポインタによると，FB25, 26Hを見ればわかります．ただし，配列領域は，ダンプにBASICプログラムを使っていると，新しい変数ができるたびにメモリの後ろの方に転送されてしまうので注意して下さい．

配列変数領域は次のようになっています．

配列変数領域

```
#D8350,83BF
8350 43 41 00 FF CA 9C 02 44 41 00 FF 00 03 41 41 00  CA.....DA....AA.
8360 24 00 01 0B 00 00 00 00 01 FF 9D 02 FD 9D 03 FA  $...............
8370 9D 04 F6 9D 05 F1 9D 06 EB 9D 07 E4 9D 08 DC 9D  ................
8380 09 D3 9D 0A C9 9D 04 42 41 00 2F 00 01 0B 00 00  .......BA./.....
8390 00 00 00 00 00 C0 00 00 00 A0 00 00 00 E0 00 00  ................
83A0 00 90 00 00 00 B0 00 00 00 D0 00 00 00 F0 00 00  ................
83B0 00 88 00 00 00 98 00 00 00 A8 00 A0 7E B7 C8 C6  ................

#P
```

なお，メモリに格納される順番は，3次元配列を例にとると次のようになります．

例　A(3，3，3)

A(0，0，0)，A(1，0，0)，A(1，1，0)………，A(2，3，3)，A(3，3，3)

次は文字列です．文字列の場合，格納されているのは実際の文字データではなく，文字列のある場所を示す，ストリング・ディスクリプタ(3バイト)です．ストリング・ディスクリプタは次のような構成になっています．

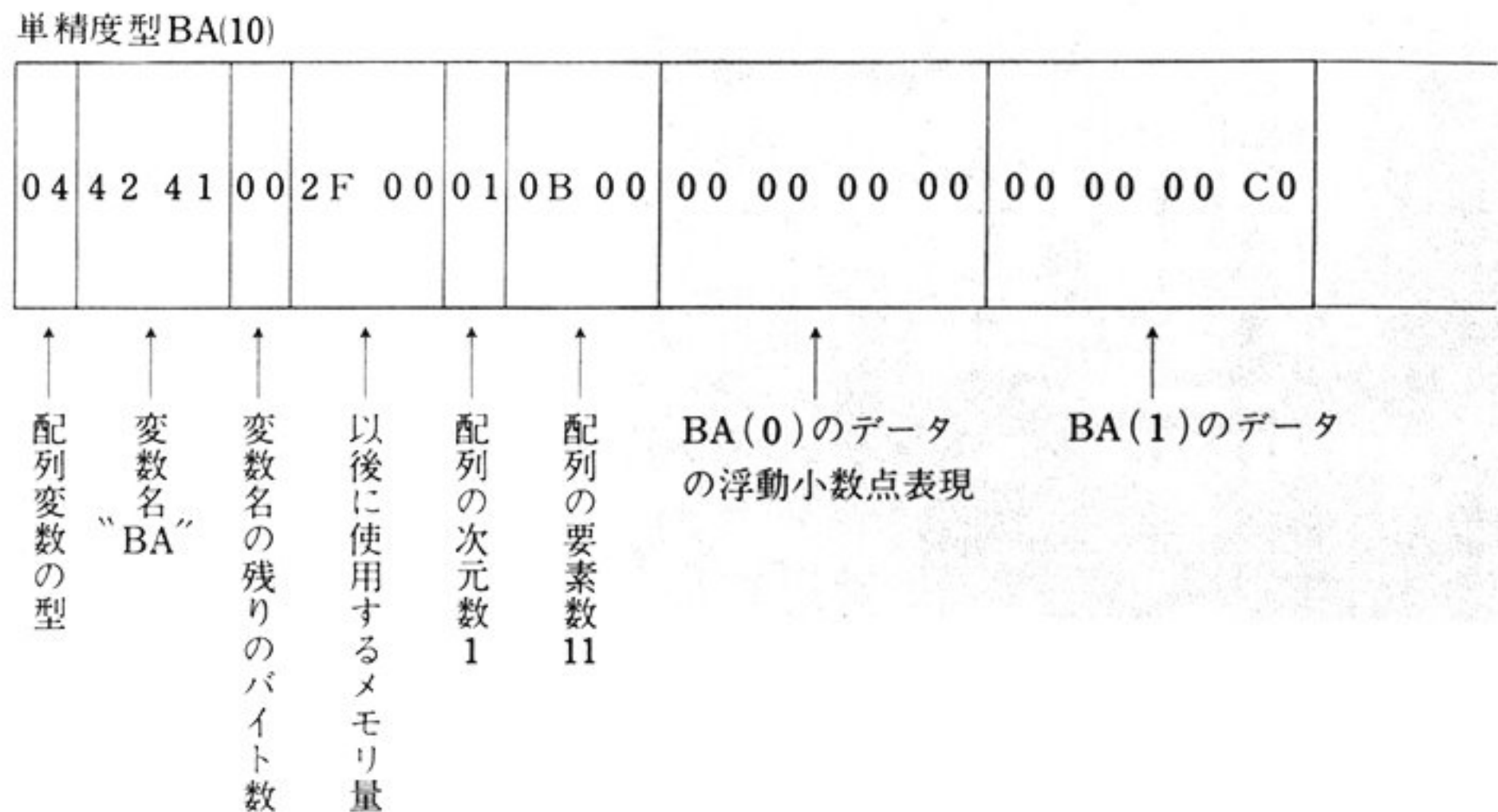



図2—1—14 配列変数の格納状態

ポインタ ：文字列が格納されているメモリの先頭アドレスを指します．

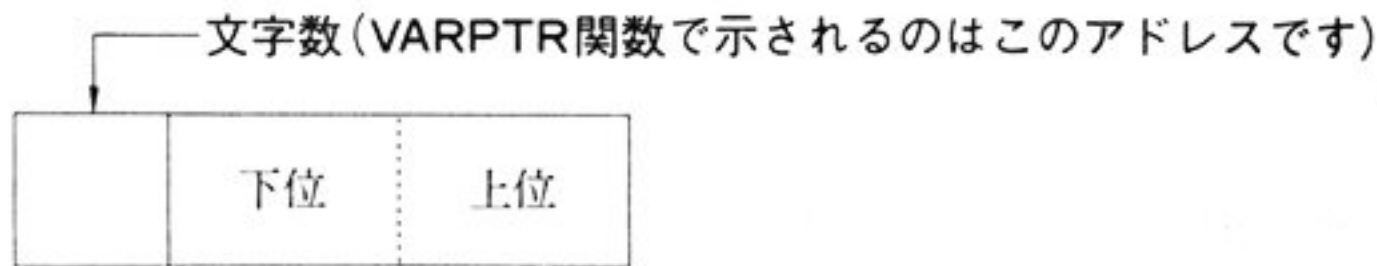



図2—1—15 ストリング・ディスクリプタの構成

文字列領域は，T-BISICでは配列領域の終わりの次のバイトからCLEARの第２パラメータで設定されるアドレスの前までとなっています．マニュアルには書かれていませんが，T-BASICではCLEARの第１パラメータは無効で，文字列領域は前記の空きエリアを自由に使っています．そのため，文字列のゴミがたまったときのガベージコレクション（エリアの整理）の時間も長く，約20～30秒かかることもあります．

2-2　内部ルーチン・ポインタを使う

巻末に，T-BASICインタプリタ（Ver 1.1）の内部ルーチンの開始アドレス一覧表を載せましたが，実際に機械語でこれらのルーチンを利用される場合，方法に困る方があると思います．

そこで，代表的ないくつかのルーチンについて，その機械語での使用法を簡単に説明します．また，BASICのポインタの応用例を紹介します．

2-2-1　内部ルーチンの使用例

内部ルーチンを使うときにはデータの受け渡しに気をつける必要があります．例えば，CIRCLEのルーチンをコールするときには，アスキーコードで示された座標や半径などのパラメータの入れてあるメモリのアドレスを，HLレジスタに入れてからコールしなければ動作をしません．

```
             *** MACHINE LANGUAGE PROGRAM ***

c060 : 21 90 d0    ;' ld hl,0d090h  ; set strings "input x="
       cd ac 61    ;' call 61ach    ; message output
       cd 58 50    ;' call 5058h    ; line input with "?"
       23          ;' inc hl        ; increment pointer
       cd 84 6e    ;' call 6e84h    ; convert ASCII CODE => FAC
       cd 6c 69    ;' call 696ch    ; csng
       cd e0 76    ;' call 76e0h    ; sin (FAC=sin(FAC))
       cd 22 70    ;' call 7022h    ; convert FAC => ASCII CODE
       e5          ;' push hl       ; save pointer of ASCII CODE
       21 a0 d0    ;' ld hl,0d0a0h  ; set strings "sin(x)="
       cd ac 61    ;' call 61ach    ; message output
       e1          ;' pop hl        ; restore pointer
       cd ac 61    ;' call 61ach    ; message output
       c9          ;' ret           ; return to BASIC

             *** DATA DUMP LIST ***

d090 : 69 6e 70 75 74 20 78 3d 00   ;' input x=
d0a0 : 73 69 6e 28 78 29 20 3d 00   ;' sin(x) =
```

写真 2-1
内部ルーチンの使用例

T-BASIC ver 1.1の内部ルーチンを使ったプログラムを次に示します．

```
1000 CLEAR ,&HBFFF
1010 CLS
1020 RESTORE
1030 WIDTH 80
1040 FOR I=1 TO 9
1050   READ A$
1060   LOCATE 20,I*2+3
1070   PRINT A$;
1080 NEXT I
1090 A$=INKEY$
1100 IF A$="" OR A$<"1" OR A$>"8" THEN 1090
1110 PRINT " "+A$;
1120 IF A$="1" THEN RESTORE 1720
```

```
1130 IF A$="2" THEN RESTORE 1830
1140 IF A$="3" THEN RESTORE 1930
1150 IF A$="4" THEN RESTORE 2040
1160 IF A$="5" THEN RESTORE 2230
1170 IF A$="6" THEN RESTORE 2300
1180 IF A$="7" THEN RESTORE 2490
1190 IF A$="8" THEN CLS:END
1200 CLS
1210 PRINT TAB(15);"XXX  MACHINE LANGUAGE PROGRAM  XXX"
1220 PRINT
1230 ROUTINE=0
1240 T=20
1250 X=0
1260 READ A$
1270 IF INSTR(A$,"/")=0 THEN 1330
1280 PRINT LEFT$(A$,4)+" : ";
1290 X=0
1300 ADDRESS=VAL("&H"+LEFT$(A$,4))
1310 A$=RIGHT$(A$,2)
1320 IF ROUTINE=0 THEN ROUTINE=ADDRESS
1330 IF LEFT$(A$,1)<>"'" THEN 1410
1340 IF RIGHT$(A$,1)="'" THEN 1380
1350 READ B$
1360 A$=A$+","+B$
1370 GOTO 1340
1380 PRINT TAB(T);";";A$
1390 X=1
1400 GOTO 1260
1410 IF A$<>"program.end" THEN 1470
1420 T=37
1430 PRINT
1440 PRINT TAB(15);"XXX  DATA DUMP LIST  XXX"
1450 PRINT
1460 GOTO 1260
1470 IF A$="data.end" THEN 1540
1480 POKE ADDRESS,VAL("&H"+A$)
1490 ADDRESS=ADDRESS+1
1500 IF X=1 THEN PRINT "        ";
1510 X=0
1520 PRINT A$+" ";
1530 GOTO 1260
1540 FOR I=1 TO 3000
1550 NEXT I
1560 PRINT CHR$(7);
1570 CALL ROUTINE
1580 GOTO 1010
1590 DATA "1 ... CIRCLE (320,100),150"
1600 DATA "2 ... PAINT (0,0),1"
1610 DATA "3 ... PSET (80,50),4"
1620 DATA "4 ... SOUND 40,100"
1630 DATA  5 ... PRINT "PASOPIA"
1640 DATA "6 ... INPUT X:PRINT SIN(X)"
1650 DATA "7 ... INPUT X:PRINT 10XX"
1660 DATA "8 ... END
1670 DATA WHICH DO YOU WANT ?
1680 '
1690 '     machine language program
1700 '
1710 '     circle
1720 DATA c000/3e,80      ,' ld a,80h      ; code of screen 2
1730 DATA       cd,22,10  ,' call 1022h    ; screen set
1740 DATA       cd,cb,08  ,' call 8cbh     ; cls
1750 DATA       21,00,d0  ,' ld hl,0d000h  ; set operand "(320,100),150"
1760 DATA       cd,a3,5a  ,' call 5aa3h    ; circle
1770 DATA       c9        ,' ret           ; return to BASIC
1780 DATA            program.end
1790 DATA d000/28,33,32,30,2c,31,30,30,29,' (320,100),150
1800 DATA       2c,31,35,30,00
1810 DATA            data.end
1820 '     paint
1830 DATA c010/3e,40      ,' ld a,40h      ; code of screen 1
1840 DATA       cd,22,10  ,' call 1022h    ; screen set
1850 DATA       cd,cb,08  ,' call 8cbh     ; cls
```

```
1860 DATA       21,10,d0   ,' ld hl,0d010h  ; set operand "(0,0),1"
1870 DATA       cd,5b,59   ,' call 595bh    ; paint
1880 DATA       c9         ,' ret           ; return to BASIC
1890 DATA           program.end
1900 DATA d010/28,30,2c,30,29,2c,31,00    ,' (0,0),1
1910 DATA           data.end
1920 '     pset
1930 DATA c020/3e,40       ,' ld a,40h      ; code of screen 1
1940 DATA       cd,22,10   ,' call 1022h    ; screen set
1950 DATA       cd,cb,08   ,' call 8cbh     ; cls
1960 DATA       21,20,d0   ,' ld hl,0d020h  ; set operand "(80,50),4"
1970 DATA       cd,bf,57   ,' call 57bfh    ; pset
1980 DATA       c9         ,' ret           ; return to BASIC
1990 DATA           program.end
2000 DATA d020/28,38,30,2c,35,30,29       ,' (80,50),4
2010 DATA       2c,34,00
2020 DATA           data.end
2030 '     sound
2040 DATA c030/21,30,d0    ,' ld hl,0d030h  ; set operand "40,10"
2050 DATA       cd,58,54   ,' call 5458h    ; sound
2060 DATA       21,40,d0   ,' ld hl,0d040h  ; set operand "42,10"
2070 DATA       cd,58,54   ,' call 5458h    ; sound
2080 DATA       21,50,d0   ,' ld hl,0d050h  ; set operand "44,10"
2090 DATA       cd,58,54   ,' call 5458h    ; sound
2100 DATA       21,60,d0   ,' ld hl,0d060h  ; set operand "45,10"
2110 DATA       cd,58,54   ,' call 5458h    ; sound
2120 DATA       21,70,d0   ,' ld hl,0d070h  ; set operand "47,10"
2130 DATA       cd,58,54   ,' call 5458h    ; sound
2140 DATA       c9         ,' ret           ; return to BASIC
2150 DATA           program.end
2160 DATA d030/34,30,2c,31,30,00          ,' 40,10
2170 DATA d040/34,32,2c,31,30,00          ,' 42,10
2180 DATA d050/34,34,2c,31,30,00          ,' 44,10
2190 DATA d060/34,35,2c,31,30,00          ,' 45,10
2200 DATA d070/34,37,2c,31,30,00          ,' 47,10
2210 DATA           data.end
2220 '     print "PASOPIA"
2230 DATA c050/21,80,d0    ,' ld hl,0d080h  ; set strings "PASOPIA"
2240 DATA       cd,ac,61   ,' call 61ach    ; message output
2250 DATA       c9         ,' ret           ; return to BASIC
2260 DATA           program.end
2270 DATA d080/50,41,53,4f,50,49,41,00    ,' PASOPIA
2280 DATA           data.end
2290 '     input x:print sin(x)
2300 DATA c060/21,90,d0    ,' ld hl,0d090h  ; set strings "input x="
2310 DATA       cd,ac,61   ,' call 61ach    ; message output
2320 DATA       cd,58,50   ,' call 5058h    ; line input with "?"
2330 DATA       23         ,' inc hl        ; increment pointer
2340 DATA       cd,04,6e   ,' call 6e04h    ; convert ASCII CODE => FAC
2350 DATA       cd,6c,69   ,' call 696ch    ; csng
2360 DATA       cd,e0,76   ,' call 76e0h    ; sin (FAC=sin(FAC))
2370 DATA       cd,22,70   ,' call 7022h    ; convert FAC => ASCII CODE
2380 DATA       e5         ,' push hl       ; save pointer of ASCII CODE
2390 DATA       21,a0,d0   ,' ld hl,0d0a0h  ; set strings "sin(x)="
2400 DATA       cd,ac,61   ,' call 61ach    ; message output
2410 DATA       e1         ,' pop hl        ; restore pointer
2420 DATA       cd,ac,61   ,' call 61ach    ; message output
2430 DATA       c9         ,' ret           ; return to BASIC
2440 DATA           program.end
2450 DATA d090/69,6e,70,75,74,20,78,3d,00,' input x=
2460 DATA d0a0/73,69,6e,28,78,29,20,3d,00,' sin(x) =
2470 DATA           data.end
2480 '     input x:print 10*x
2490 DATA c090/21,b0,d0    ,' ld hl,0d0b0h  ; set strings "input x="
2500 DATA       cd,ac,61   ,' call 61ach    ; message output
2510 DATA       cd,58,50   ,' call 5058h    ; line input with "?"
2520 DATA       23         ,' inc hl        ; increment pointer
2530 DATA       cd,04,6e   ,' call 6e04h    ; convert ASCII CODE => FAC
2540 DATA       cd,f8,68   ,' call 68f8h    ; cint
2550 DATA       2a,44,fc   ,' ld hl,(0fc44h); set hl register pair = FAC
2560 DATA       11,0a,00   ,' ld de,000ah   ; set de register pair = 10
2570 DATA       cd,b2,6a   ,' call 6ab2h    ; integer multiplication
2580 DATA       cd,6c,69   ,' call 696ch    ; csng
```

```
2590 DATA      cd,22,70   :' call 7022h     ; convert FAC => ASCII CODE
2600 DATA      e5         ,' push hl        ; save pointer of ASCII CODE
2610 DATA      21,c0,d0   ,' ld hl,0d0c0h   ; set strings "x*10="
2620 DATA      cd,ac,61   ,' call 61ach     ; message output
2630 DATA      e1         ,' pop hl         ; restore pointer
2640 DATA      cd,ac,61   ,' call 61ach     ; message output
2650 DATA      c9         ,' ret            ; return to BASIC
2660 DATA         program.end
2670 DATA d0b0/69,6e,70,75,74,20,78,3d,00,' input x=
2680 DATA d0c0/78,2a,31,30,20,3d,00       ,' x*10 =
2690 DATA         data.end
```

実行させると解かるように，このプログラムは，メニューに示した各ルーチンに対して必要なパラメータを設定し，受け渡しを行ったのち，そのルーチンをコールするもので，ほとんどのルーチンは，このメニューのいずれかの方法を応用すれば使用できます．

以下にプログラムの詳細を示します．

プログラムの説明

1710～1810　SCREENモード２とするために，アキュムレータに80Hをロードしてスクリーンのルーチンをコールしています．円を描くためのパラメータの(320, 100), 150はアスキーコードでD000Hから書き込まれていますから，アドレスをHLレジスタにロードしてCIRCLEのルーチンをコールします．また，パラメータとして書き込まれたデータの終わりには必ず00Hを書き込む必要があります．

1820～1910　PAINTのルーチンをコールするための機械語プログラムとパラメータのデータです．PAINTの場合も同様にアスキーコードで格納されたデータのアドレスをHLレジスタにロードしてからPAINTのルーチンをコールしなければなりません．

1920～2020　PSETのルーチンをコールするための機械語プログラムとパラメータのデータです．

2030～2140　SOUNDのルーチンをコールするための機械語プログラムとパラメータのデータです．

2220～2280　"PASOPIA"を画面に表示するために文字列PRINTのルーチンをコールしています．表示するデータはアスキーコードで示され，終わりには00Hをつけておきます．これも開始アドレスをHLレジスタにロードしてから，文字例PRINTのルーチンをコールします．

2290～2470　Xを入力してSIN(X)を出力するプログラムです．HLレジスタをインクリメント（１つ増やす）しているのは，識別コードがHLレジスタの示すアドレスにあるからです．

2490～2690　整数のかけ算です．68F8Hからは単精度変換のサブルーチンになっています．PUSHでスタックにレジスタを退避しているのは，CRT出力ルーチンにデータを送るためにHLレジスタを使うからです．

2-2-2　RAMを32K増やす(未使用RAMの活用)

ROMBASIC起動時はRAMの半分にあたる32キロバイトが使われていません．この未使用RAM(裏RAM)をデータ・エリアとして使用するプログラムを紹介します．

```
1000 '***
1010 '***    RAMFILE for T-( rom ) BASIC
1020 '***
1030 '
1040 '*
1050 '* SET M-SUB
1060 '*
1070 CLEAR,&HDFFF
1080 RAMFILE=&HF783   :' M-SUB ADR
1090 RFWRITE=&HF783   :' ADR (WRITE)
1100 RFREAD =&HF794   :' ADR (READ)
1110 CELLNO =&HF780   :' ADR (CELL NO SET)
1120 BUFFER =&HF781   :' ADR (BUFFER START ADRESS)
1130 ADR=RAMFILE
1140 RESTORE 1710
1150 READ A$
1160 WHILE A$<>"END"
1170   POKE ADR,VAL("&H"+A$)
1180   ADR=ADR+1
1190   READ A$
1200 WEND
1210 '*
1220 '* GET ADR FILE BUFFER#0
1230 '*
1240 FIELD#0,1 AS A$
1250 POKE BUFFER  ,PEEK(VARPTR(A$)+1)
1260 POKE BUFFER+1,PEEK(VARPTR(A$)+2)
1270 '*
1280 '* SAMPLE
1290 '*
1300 WIDTH 80:SCREEN 1:COLOR 7,0:KEY OFF:CLS
1305 PRINT" PASOPIA RAM FILE SAMPLE PROGRAM"
1310 FIELD#0,128 AS A1$,128 AS A2$
1320 LSET A1$=STRING$(128,0)
1330 LSET A2$=STRING$(128,0)
1340 PRINT"ﾀﾀﾞｲﾏ ram ｦ ｼｮｷｶ ﾁｭｳﾃﾞｽ"
1350 FOR I=0 TO 127    :' clear ramfile.
1360   POKE CELLNO,I
1370   CALL RFWRITE
1380 NEXT I
1390 '*
1400 '* INPUT FOR READ/WRITE
1410 '*
1430 PRINT
1440 INPUT"READ OR WRITE ( R / W)";C$
1450 IF C$="R" OR C$="r" THEN GOSUB 1490:GOTO 1440
1460 IF C$="W" OR C$="w" THEN GOSUB 1600:GOTO 1440
1470 IF C$="E" OR C$="e" THEN END
1480 GOTO 1440
1490 '*
1500 '* READ SUB
1510 '*
1520 INPUT "CELL NO ";CELL
1530 IF CELL<0 OR CELL>127 THEN PRINT"ILLEGAL ":RETURN
1540 POKE CELLNO,CELL
1550 CALL RFREAD
1560 PRINT"RECORD NO ";CELL
1570 PRINT A1$
1580 PRINT
1590 RETURN
1600 '*
1610 '* WRITE SUB
1620 '*
```

```
1630 INPUT "CELL NO ";CELL
1640 IF CELL<0 OR CELL>127 THEN PRINT"ILLEGAL":RETURN
1650 INPUT "DATA:";D$
1660 LSET A1$=D$
1670 POKE CELLNO,CELL
1680 CALL RFWRITE
1690 LSET A1$=STRING$(128,0):GOTO 1540
1700 RETURN
1710 '**** M-SUB ************************************************************
1720 'WRITE SUB
1730 DATA 3A,80,F7,2A,81,F7,57,1E,00,43,0E,FF,F3,ED,B0,FB,C9
1740 'READ SUB
1750 DATA 3A,80,F7,ED,5B,81,F7,67,2E,00,45,0E,FF,3E,02,F3,D3,3C,ED,B0,AF,D3,3C,F
B,C9
1760 DATA END
```

このプログラムは，裏RAMの32キロバイトを256バイトずつの128個のエリアに分割し，F780Hからの機械語サブルーチンでバンク切換とブロック転送を行うものです．

RAMに書き込むときには，F780H番地に書き込むエリアをPOKEしてF783Hをコールします．コールするとファイルバッファ0に書き込まれたメモリをF780H番地で指定したエリアにブロック転送します．

RAMから読み出したければF780H番地でエリアの番号を指定してF794Hをコールします．このときには書き込むときと逆のブロック転送が行われます．

ファイルバッファ#0を使用しているので，FIELD命令で使用する文字数を設定することができ，BASICプログラムの処理速度を速めています．

2-2-3 DISK-BASICを切り離す方法

BASICのプログラムをメモリ内に残したままDISK-BASICを切り離す方法を紹介します．次の手順で操作して下さい．

```
LIST
10 REM DISK BASIC TO ROM BASIC SAMPL
E
20 PRINT "THIS IS SAMPLE PROGRAM"
30 END
Ok
?PEEK(&H2769)   ┐
 72             │プログラム領域開始のポインタを調べる。┐
Ok              │                                      │
?PEEK(&H276A)   ┘                                      │
 154                                                   │プログラムが格納されている場所を調べる。
Ok                                                     │
?PEEK(&H2A36)   ┐                                      │
 148            │変数領域の開始ポインタを調べる。      ┘
Ok              │
?PEEK(&h2A37)   ┘
 154
Ok
```

1) プログラム領域開始ポインタの値を調べる．

DISK版では2769Hに下位，276AHに上位のアドレスが格納されているのでここをPEEKで調べます．下位，上位

の内容を調べたら紙にメモして下さい.

2) 変数領域の開始アドレスを調べる. ポインタは2736Hと2737Hにあります.

3) ディスクユニットをOFFにしてリセットスイッチを押す.

4) ROMBASICのポインタに書き込む.

```
How many files(0-15)? 0
Toshiba T-BASIC Ver 1.1
(c) 1982 by Microsoft
29600 Bytes free
Ok
poke &hf857,72      ]
Ok                  ] プログラム領域開始ポインタに書込む
poke &hf858,154     ]
Ok
poke &hfb23,148     ]
Ok                  ] 変数領域開始ポインタに書込む
poke &hfb24,154     ]
Ok
clear               ← 他のポインタも書換られる
Ok
LIST
10 REM DISK BASIC TO ROM BASIC SAMPL
E                                       ← 復活する
20 PRINT "THIS IS SAMPLE PROGRAM"
30 END
Ok
```

以上の操作で, プログラムを残したまま, ROMBASICを立ち上げることができます.

2-2-4 UNLIST・UNSAVE

何かの都合でプログラムをメモリ中に残して席を離れるときに, プログラムをコピーされてしまうことがわします. パリピアではRAMPACを使えば瞬時にSAVEできるため, 何らかの対策をとりたいところです. そのために, 実行はできるがSAVEはできず, LISTもとれないようにする方法があります.

次のプログラムでは, 一番最初の行番号をFFFFHに書き変えてUNLISTにしています. UNSAVEはリンクポインタを書き変えてBASICインタプリタに, プログラムが入っていない, と感違いさせることによって実現しています.

```
10 '***
20 '*** UNLIST & UNSAVE
30 '***
40 INPUT"Do you use floopy-disk (Y/N)";A$
50 IF A$="Y" OR A$="y" THEN TXTTAB=&H2769 ELSE TXTTAB=&HF857
60 PRINT"Passward:"
70 PASS$=INPUT$(3)
80 LOW=PEEK(TXTTAB):HIGH=PEEK(TXTTAB+1)
90 ADR=LOW+HIGH*256+2
95 '------ PASSWARD CHECK --------------------------------------------------
100 IF PASS$="AAA" THEN 160
110 IF PASS$="BBB" THEN 240
120 IF PASS$<>"UNL" THEN PRINT CHR$(7);"Passward ｶﾞ ﾁｶﾞｳ!!!":END
125 '------- UNLIST RECOVER -------------------------------------------------
```

```
130 POKE ADR,&HA
140 POKE ADR+1,0
150 PRINT"recover ok!":END
160 '------- UNLIST -----------------------------------------------------
170 POKE ADR,&HFF:POKE ADR+1,&HFF
180 PRINT"unlist ok!"
190 END
200 '------- UNSAVE & UNLIST --------------------------------------------
210 'If you want recover then you must type following command.
220 '          ------------->  'POKE (n1),(n2):POKE (n3,n4)[RETURN]'
230 '
240 A1=PEEK(ADR-2)
250 A2=PEEK(ADR-1):PRINT ADR-2,A1,ADR-1,A2
255 '                       ^n1   ^n2  ^n3 ^n4
260 POKE ADR-2,0
270 POKE ADR-1,0
280 PRINT"UNSAVE OK!"
290 END
```

このプログラムは次のように操作します.

まず,目的のプログラムの後ろに,上のプログラムを打ち込んで,RUN60000として下さい.

・UNLIST化するとき

DISKを使っているかと聞いてくるので,使っていればYと答え,使っていないときはNと答えて下さい.続いてパスワードを聞かれますので,好きな文字列をタイプして下さい.パスワードのエコーバック(タイプした文字がCRTに表示されること)はないので注意して下さい.パスワードがあっていれば,unlist ok!と表示されます,

・UNLISTを解除したいとき

GOTO60000とし,前に入力したパスワードを入力して下さい.

・UNSAVE化するとき パスワードを入力します.ただし,UNSAVE化したときはSAVEできない代わりにLISTもとれず,RUNもしません.実行時に数字が4つのCRTに出力されるので,これを記録しておいて下さい.

・UNSAVEを解除するとき 前に出力された数字を順にn1~n4として,ダイレクトで次の命令を実行して下さい.

```
POKE n1, n2 : POKE n3, n4
```

これで復活します.

今回用いた方法を説明しましょう.UNLIST化するために最初の行番号が格納されるアドレス2バイトにFFHを書き込んでいます.こうすると,インタプリタは勝手に以後にプログラムはないものと解釈してLISTしなくなるのです.ただし,このままではSAVEは可能です.なぜならSAVEするときにはプログラム領域開始のポインタの示すアドレスから変数領域開始のポインタの示すアドレスまでをSAVEするためです.さらに,RENUMをかけると,GOTOなどの飛び先は直されないのでエラーが出ますが,プログラムは回復します.(ただし飛び先は狂っています),

UNSAVEはリンクポインタを書き変えてインタプリタにプログラムが入っていないものと解釈させることにより実現していますので実行できません.

2-2-5　プログラムのAPPEND

フロッピィディスク使用時やRAMPAC 2 使用時は複数のプログラムの結合命令としてMERGEを使うことができるのですが，カセットベースで使用しているときはMERGEを使うことができず，汎用性のあるサブルーチンを作ってもキーボードから打ち込まなければなりません．しかし，カセットでのMERGEはポインタの書き換えだけで実現することができるのです．

BASICのプログラムに関係するポインタをいじるので，手動で行う以外に方法はありませんが，アペンド(プログラムの結合)することができます．

順を追って方法を説明しましょう．

1)まず前に置くべきプログラムを入力して下さい．

2)プログラム領域開始ポインタ(F857H～F858H)の値を調べ記録して下さい．

3)変数領域開始ポインタの値を調べ，その値によって示されるアドレスから2を引いた値を，プログラム領域開始ポインタに書き込んで下さい．

4)後ろに結合されるプログラムをCLOADして下さい．このプログラムの開始行番号は，前のプログラムの終わりの行番号より大きくなるようにして下さい．

5)プログラム領域開始ポインタに最初の値を書き込めばでき上りです．

次に例を示します．

```
How many files(0-15)? 0
Toshiba T-BASIC Ver 1.1
(c) 1982 by Microsoft
29600 Bytes free
Ok
10 '***
20 '***  Append sample 1
30 '***
40 PRINT"This is program 1"
50 END
Ok
?PEEK(&hF857),PEEK(&hF858)           ← プログラム領域開始のポインタを調べる。
 89           130
Ok
?PEEK(&hFB23),PEEK(&hFB24)           ← 変数領域開始のポインタを調べる
 172          130
Ok
POKE &HF857,170:POKE &HF858,130 ← プログラム領域開始ポインタを書換える
Ok
LIST
Ok


CLOAD
Found:sam2                          ← 後につけるプログラムをロードする
Ok
LIST
1000 '***
1010 '***  Append sample 2
1020 '***
1030 PRINT"This is program 2"  ← 後につくプログラムのLIST
1040 END
Ok
POKE &HF857,89:POKE &HF858,130 ← プログラム領域開始ポインタを元の値にする
Ok
```

```
LIST
10 '***
20 '***  Append sample 1
30 '***
40 PRINT"This is program 1"
50 END
1000 '***
1010 '***  Append sample 2
1020 '***
1030 PRINT"This is program 2"
1040 END
OK
```

←プログラムが連結した

では，実際にメモリ上ではどのようなことが起こるのでしょうか．

アペンドする前のプログラム1の終わりの部分は次のようになっています．

```
#D8280,82AF
8280 8B 82 1E 00 3A 8F E5 2A 2A 2A 00 A4 82 28 00 91   ....:..***...(..
8290 22 54 68 69 73 20 69 73 20 70 72 6F 67 72 61 6D   "This is program
82A0 20 31 22 00 AA 82 32 00 81 00 00 00 1E E5 84 42    1"...2........B

#P
```

プログラムの終りには00Hが3つ並んでいる。
この2つめの番地からテープを読込ませる。

最後の行のリンクポインタが示すアドレスからアペンドしたいプログラムを読み込ませればよいのです．

```
#D8280,82AF
8280 8B 82 1E 00 3A 8F E5 2A 2A 2A 00 A4 82 28 00 91   ....:..***...(..
8290 22 54 68 69 73 20 69 73 20 70 72 6F 67 72 61 6D   "This is program
82A0 20 31 22 00 AA 82 32 00 81 00 B5 82 E8 03 3A 8F    1"...2.......:.

#P
```

アペンド後は，先ほどのリンクポインタの示すアドレスにプログラム2のリンクポインタが格納されます．

2-2-6 メモリをALL-RAMに

ROMBASIC使用時は，64キロバイトあるメモリのうち半分にあたる32キロバイトのRAMは使われていません．次のプログラムを実行するとBASIC-ROMの内容がRAMにコピーされます．

```
10 ADR=&HD000
20 FOR I=0 TO 19
30 READ A$
40 POKE ADR+I,VAL("&H"+A$)
50 NEXT
60 CALL ADR
70 END
100 DATA 21,00,00         :' LD HL,0000H
110 DATA 11,00,00         :' LD DE,0000H
120 DATA 01,FF,7F         :' LD BC,7FFFH
130 DATA F3               :' DI
140 DATA ED,B0            :' LDIR
150 DATA FB               :' EI
```

```
160 DATA 0E,3C            :' LD C,3CH
170 DATA 3E,02            :' LD A,02H
180 DATA ED,79            :' OUT (C),A
190 DATA C9               :' RET
```

このプログラムは，ROMの内容を同じアドレスのRAMに対しブロック転送で書き込んだ後，バンク切換を行わせています．RAMが切り離されているときでも，書き込むことはできるためこのようなことが可能です．

2-2-7　プログラムの回復法

NEW命令を投入したりリセットをかけたりしたときにはプログラムは消えますが，実はメモリ上では書き込まれたすべてのメモリがクリアされるわけではなく，一部のポインタのみが書き換られているのに過ぎません．書き換えられたポインタを元に戻せばプログラムを回復することができます．この手順を示しましょう．

・変数領域開始のポインタ(ROM版FB23H)をPOKEで書き換え，変数を使用してもメモリを破壊されないようにする．書き換えたらCLEAR命令を実行する．

・テキスト開始ポインタを調べる．(テキスト開始のポインタはROM版ではF857Hにあります)．

・ダイレクトでテキスト番地からメモリの内容をダンプして，最初のリンクポインタを見つけ書き換える，また，00が3個続いているところを見つけ，その2番目の00のアドレスを変数領域開始のポインタに書き込む．

・CLEARを実行する．

以上の手順でプログラムが回復するのですが，とても手間がかかります．次の機械語サブルーチンを使えば比較的簡単にプログラムを復活することができます．(このプログラムはROM版Ver 1.1用です)．

```
F000 2A 57 F8   LD   HL,(F857H)   ; LOAD  TEXT START
F003 36 01      LD   (HL),01H     ; ﾘﾝｸ ﾎﾟｲﾝﾀ SET
F005 CD B0 1F   CALL 1FB0H        ; ﾘﾝｸ ﾎﾟｲﾝﾀ SET CALL
F008 CD F2 1F   CALL 1FF2H        ; ｱﾄﾞﾚｽ -> ｷﾞｮｳ
F00B 23         INC  HL
F00C 23         INC  HL
F00D 22 23 FB   LD   (FB23H),HL   ; VAL POINTER SET
F010 22 25 FB   LD   (FB25H),HL   ; ARRAY POINTER SET
F013 C9         RET
```

このプログラムを使用するときは必ず次のようにUSRでコールして下さい．

DFF　USR1=&HF000：USR1(0)

CALLでは変数をオペランドとしているため，変数が宣言されることによりプログラムエリアが破壊されてしまうので使わないで下さい．

なお，機械語プログラム中で使用している内部サブルーチンについては，APPENDIXをご覧下さい．

第3章

グラフィック

第3章　グラフィック

3-1　SCREEN命令とV-RAM

パソピアは16キロバイトのV-RAMを持ち，1章でも述べたようにI/O経由で使用し，8255により制御しています．

キャラクタは247種ありますが，T-BASICでは19H以下のコードの文字を扱うことができないため，実際に使えるのは216種類です．キャラクタは8×8ドットで表され，CG(キャラクタ・ジェネレータ)から供給されます．

表示するスクリーンモードは3種類で，テキストモード，グラフィックモード，ファイングラフィックモードとなっています．

これらのモードの切換えは，8255のポートをアクセスすることによって行います．

次ページ，図3-1-1にコントロール内容を示します．

各モードを比較してみましょう．

・SCREEN0(テキストモード)

テキストモードでは，すべてのディスプレイコードをキャラクタ・ジェネレータで文字パターンに変換します．この場合，V-RAMには次のように書き込まれます．

図3—1—2　ディスプレイコード・モード0

このとき，ビット8が0でビット3～7が1のときにはカラーアトリビュート・キャラクタであると判断され，その行の後にあるキャラクタがビット0～2まで示される色に着色されます．

#	ポート	動作モード及び ポート名	端子	アクティブ	コントロール内容
8255−1	A	出力 0 8 H	PA 7	H	ファイングラフィックモードの設定
			PA 6	H	グラフィックモードの設定
			PA 5	H	80桁表示モードの設定(CRTのみ)
			PA 4		
			PA 3		
			PA 2	H	背景色の緑信号を出力します.
			PA 1	H	背景色の赤信号を出力します.
			PA 0	H	背景色の青信号を出力します.
	B	入力 0 9 H	PB 7	H	V-RAMの読み出しデータ(2^8)を入力します.
			PB 6	H	V-RAMアクセス呼のビジィー信号を入力します
			PB 5	H	V-SYNC信号を入力します
			PB 4	H	表示装置の種別入力(CRT="1", LCD="0")
			PB 3 〜 PB 0		
	C	出力 0 A H	PC 7	H	V-RAMの書き込みデータ(2^8)を出力します.
			PC 6	L	V-RAMのR/W信号を設定(Read="1", Write="0")
			PC 5		
			PC 4 〜 PC 0	H	V-RAMアドレスの2^{13}(PC 5)〜2^8(PC 0)を出力します.
8255−2	A	出力 0 0 H	PA 7 〜 PA 0	H	V-RAMアドレスの 2^7(PA 7)〜 2^0(PA 0)を出力します.
	B	出力 0 1 H	PB 7 〜 PB 0	H	V-RAMの書き込みデータを出力します. PB 7 = 2^7……PB 0 = 2^0
	C	入力 0 2 H	PC 7 〜 PC 0	H	V-RAMの読み出しデータを入力します. PC 7 = 2^7……PC 0 = 2^0

○表示モード/サイズ制御

8255で設定する3種の表示モード及びCRTの36/80行表示の制御を行う.

○背景色制御

ソフト・ウェアから指定された背景色を8255を介して設定する.

背景色の設定は表示モードの設定と同一の出力ポートであるので, メイン・メモリ(RAM)内に常に記憶している.

図3−1−1　8255のコントロール内容

またビット８が１であればその１文字が文字色と背景色を反転して表示されます.（写真３－１）
次のプログラムはキャラクタの反転表示の例です.

```
10 '***
20 '***  REVERSE
30 '***
100 WIDTH 36:SCREEN 2:CLS
110 OUT 8,16    :'FOR TEXT MODE SET
120 LOCATE 0,0:PRINT"** REVERSE DISPLAY SAMPLE ***"
130 DIM BIT(7)
140 FOR I=0 TO 7
150   BIT(I)=2^I
160 NEXT
170 X=0:Y=10
180 FOR I=1 TO 255
190   FOR J=7 TO 0 STEP -1
200     IF I AND BIT(J) THEN PSET(X*8+7-J,Y*8)
210   NEXT
220 X=X+1
230 IF X=36 THEN X=0:Y=Y+1
240 NEXT I
```

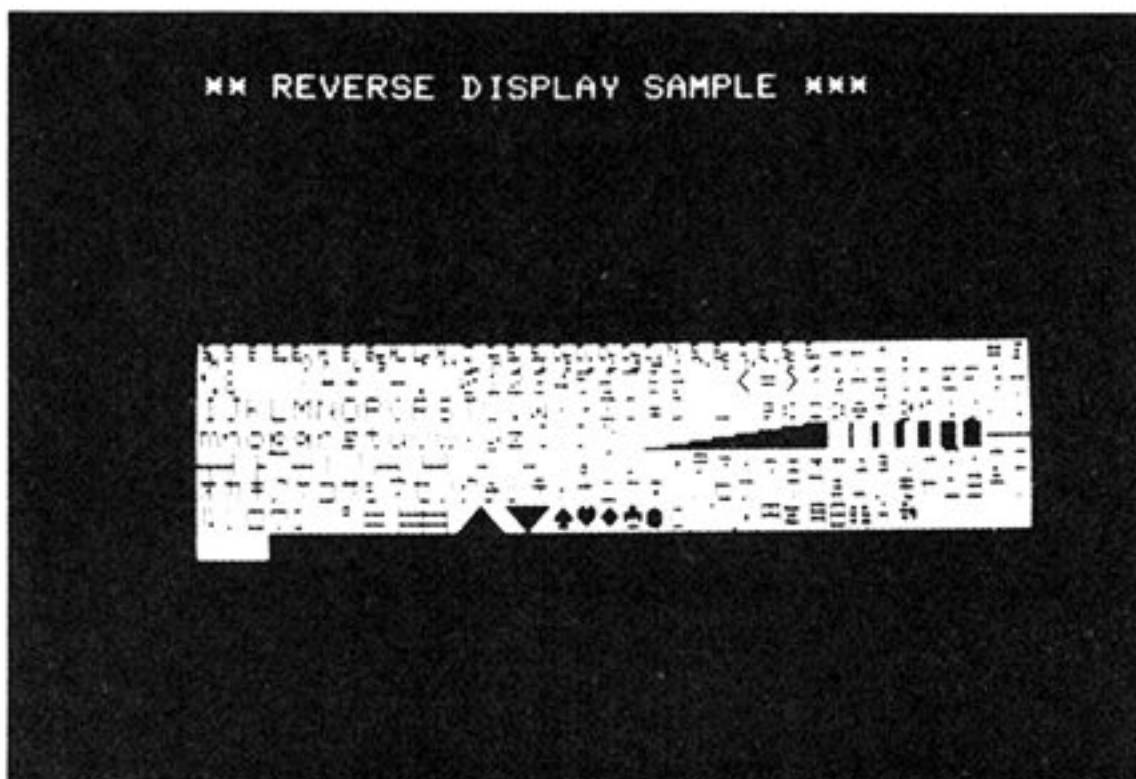



写真３－１
反転の表示

110行のOUT命令はテキストモードの指定を行っています.100行のSCREEN命令は,BASICでディスプレイコードを出力させるために必要です.

・SCREEN1(グラフィックモード)

このモードでは，各ドットごとに８色指定できます．このとき，ディスプレイコードは次ページ図３－１－３のようになります.

このモードでは，ビット８はテキストとグラフィックの区別に使われています．このため，反転表示はできません.

グラフィックデータは，ビット６～４が左側のドット，ビット２～０が右側のドットの色を指定しています.ドット表示はPSET等の命令で行われていますが,片方のみ色を指定したときには,もう片方は背景色が書き込まれます．このとき，全体の背景色が変っても，以前のドットの背景色を新しい背景色に換えるような処置はなされません.

色指定のアトリビュート・キャラクタはテキストデータに対してのみ作用します.

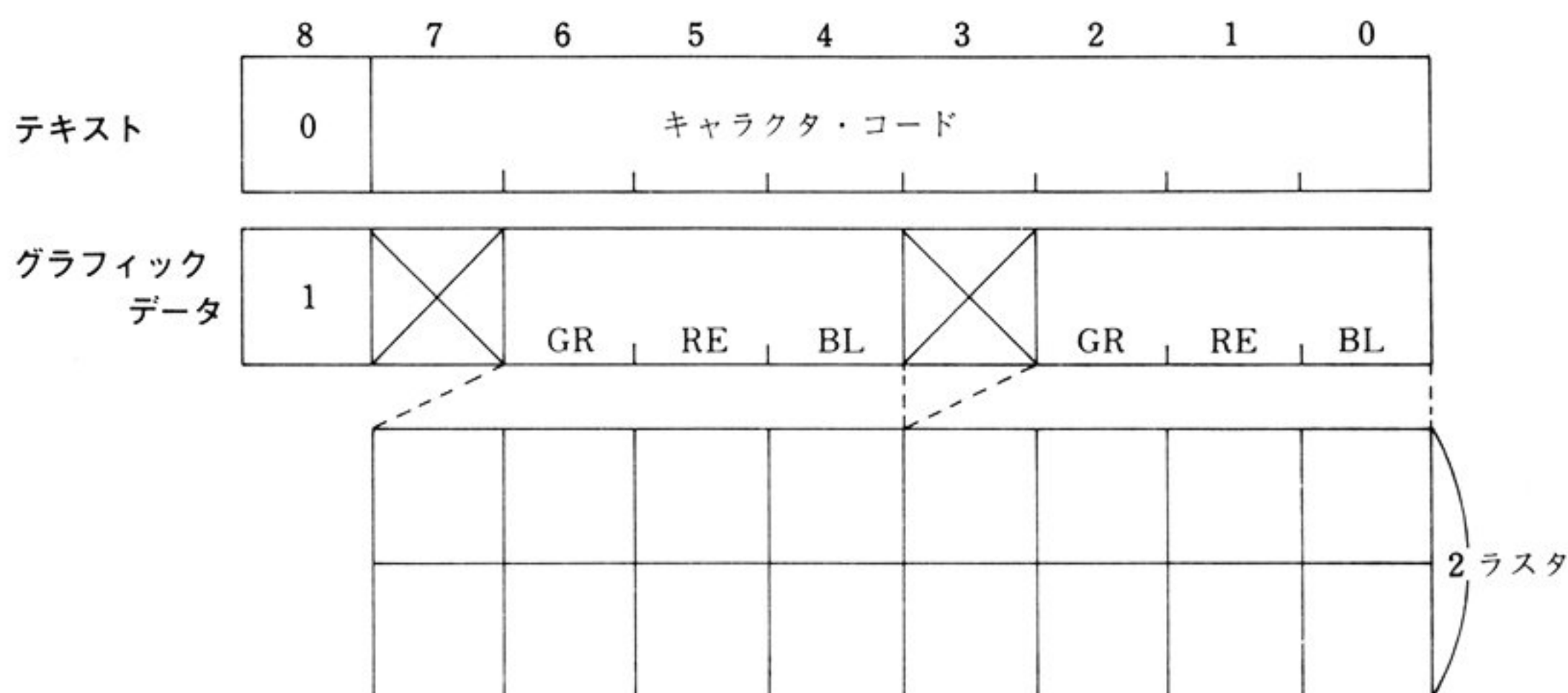



図3－1－3　ディスプレイコード・モード1

・SCREEN2(ファイン・グラフィックモード)

最高640×200の分解能をもち，横方向8ドット単位で色指定が可能です．しかし，T-BASICで色指定するためには，カラーアトリビュート・キャラクタが8×8ドット分を占有してしまうため少々難しくなります．このモードでは，ディスプレイコードは次のようになります．

図3－1－4　ディスプレイコード・モード2

縦方向1ドットごとの着色は，カラーキャラクタを並べ，不要分をLINEもしくはPSETで消すことによって可能です．

```
10 '***
20 '***    ｶﾗｰ ｱﾄﾘﾋﾞｭｰﾄ ｷｬﾗｸﾀｰ
30 '***
100 WIDTH 80:COLOR 7,0:SCREEN 2:CLS
110 DIM COL$(7)
120 FOR I=0 TO 7
130   COL$(I)=CHR$(&HF8+I)
140 NEXT I
145 PRINT TAB(30);" << COLOR SAMPLE >>"
150 FOR I=33 TO 247
160   PRINT COL$(I MOD 8);CHR$(I);
```

```
170 NEXT I
180 PRINT
190 FOR I=7 TO 1 STEP -1
200   X0=I*80-80
210   LINE(X0+8,100)-(X0+79,200),,BF
220   FOR J=12 TO 24
230     LOCATE X0¥8,J:PRINT COL$(I);
240   NEXT J
250 NEXT I
260 LOCATE 0,11
270 END
```

3-2　アトリビュート・キャラクタ

T-BASICでは，テキストやSCREEN2のファイン・グラフィックの色指定は基本的には横の行ごと(8ドットごと)になっています．これは，横方向36字もしくは80字の画面の左側にカラー・アトリビュート・キャラクタが書き込まれているからです．

1行に複数の色を使いたいときは，着色したいキャラクタの前にアトリビュート・キャラクタを書き込むことによって可能です．

3-2-1　1つのキャラクタを複数色で

アトリビュート・キャラクタをうまく使うと，1つのキャラクタを複数色で表示することができます．アトリビュート・キャラクタを書き込んだときに，必要のない部分をLINEやPSETで消してしまえばよいのです．

次のプログラムでは，縦方向1ドット単位で色を書き換えています．行番号190～250で不要な部分のアトリビュート・キャラクタをLINE命令で消しています．(カラーページ①参照)

```
10 '***
20 '***  SCREEN 2 COLOR SAMPLE
30 '***
100 WIDTH 80:SCREEN 2:CLS:KEY OFF
110 '
120 PRINT TAB(25);" SCREEN 2 COLOR SAMPLE":PRINT
130 FOR I=1 TO 8
140   PRINT TAB(20);
150   FOR J=32*I TO 32*I+32
160     IF J<248 THEN PRINT CHR$(J);
170   NEXT J
180   PRINT
190 NEXT I
200 LINE(160,100)-(639,150),,BF
210 '*
220 '* SET COLOR CHAR.
230 '*
240 FOR I=1 TO 24
250   FOR J=1 TO 7
260     LOCATE J,I:PRINT CHR$(248+J);
270   NEXT J
280 NEXT I
290 FOR J=1 TO 199
300   LINE((J MOD 7)*8,J)-(64,J),0
310 NEXT J
320 LOCATE 0,0
```

3-2-2　アトリビュート・キャラクタで高速グラフィックを使う

SCREEN 2では，アトリビュート・キャラクタを使うことによって，図形などをえがきなおすことなく，高速に色を変えることができます．画面に1度描いておき，その前に置いたアトリビュート・キャラクタを書き換えることによって，簡単なアニメーションのようなものを作ってみました．

```
10 '**
20 '** color demo for PA7010
30 '**        (T-BASIC)
40 '*
50 '*   START
60 '*
70 CLEAR: RANDOMIZE TIME/4
80 SCREEN 2:WIDTH 80:OUT 8,160:COLOR 0,0:CLS
90 DEFINT A-Z
100 DIM COL$(8)
110 FOR I=0 TO 7
120      COL$(I)=CHR$(248+I)
130 NEXT I
140 COL$(8)=" "
150 GOSUB 1150
160 GOSUB 850
170 GOSUB 1250
180 GOSUB 1020
190 '*
200 '*  DEMO START
210 '*
220 FOR LOOP=1 TO 10
230 ' CHANGE COLOR ( ALL CIRCLE )
240 COLOR 0
250 FOR COL=0 TO 7
260     FOR CN=1 TO 10
270         GOSUB 770
280     NEXT CN
290 NEXT COL
300 COLOR 7,0
310 COL=0:FOR CN=1 TO 10:GOSUB 770:NEXT
320 'CHAGE COLOR (2 CIRCLE)
330 FOR COL0=1 TO 7
340     FOR CN0=1 TO 10
350         COL=(COL0-1) MOD 8
360         CN=(CN0+8) MOD 10+1:GOSUB 770
370         COL=COL0
380         CN=CN0:GOSUB 770
390         COL=(COL0-1) MOD 8
400         CN=(CN0+4) MOD 10+1:GOSUB 770
410         COL=COL0
420         CN=(CN0+5) MOD 10+1:GOSUB 770
430     NEXT CN0
440 NEXT COL0
450 GOSUB 1020
460 ' CHANGE COLOR (1 CIRCLE )
470 FOR K=0 TO  7
480     FOR L=0 TO 10
490         CN0=(K+L) MOD 10+1
500         COL=(K+L) MOD 7+1
510         CN=CN0
520         GOSUB 770
530         CN=(CN0+9) MOD 10+1
540         COL=0
550         GOSUB 770
560     NEXT L
570 NEXT K
580 ' ALL CIRCLE
590 FOR COL=0 TO 7
```

```
600     FOR CN=1 TO 10
610         GOSUB 770
620     NEXT CN
630 NEXT COL
640 ' CHANGE BORDER COLOR
650 FOR COL=1 TO 7
660    COLOR RND*6+1:FOR Y=0 TO 4:LOCATE 0,Y:NEXT
670    COLOR 7  ,(COL+1) MOD 8
680    'FOR I=1 TO 1000:NEXT
690     FOR CN=1 TO 10
700         GOSUB 770
710     NEXT CN
720     FOR Y=1 TO 1000:NEXT
730 NEXT COL
740 COLOR ,0
750 NEXT LOOP
760 END
770 '*
780 '* SET COLOR FOR CIRCLE
790 '*
800 ' INPUT ...CN  AS CIRCLE NUMBER
810 '           COL AS COLOR
820     LOCATE XPOS(CN),YPOS(CN)
830     PRINT CHARDATA$(CN,COL);
840 RETURN
850 '*
860 '*  DRAW CIRCLE
870 '*
880 PI!=3.14159
890 FOR I!=0 TO PI!*2 STEP PI!/ 5
900     X!=COS(I!)*100+320
910     Y!=SIN(I!)*50+100
920     CIRCLE(X!,Y!),20,1
930 NEXT
940 FOR I!=0 TO PI!*2 STEP PI!/ 5
950     X!=COS(I!)*100+320
960     Y!=SIN(I!)*50+100
970     PAINT(X!,Y!),1,1
980 NEXT
990 RETURN
1000 '
1010 '
1020 '*
1030 '* SCREEN CLEAR ( DAMMY )
1040 '*
1050     COLOR 0
1060 FOR I=5 TO 24
1070     LOCATE 0,I
1080 NEXT I
1090 FOR I=1 TO 10
1100         LOCATE XPOS(I),YPOS(I)
1110         PRINT CHARDATA$(I,0)
1120 NEXT I
1130 COLOR 7
1140 RETURN
1150 '*
1160 '* WRITE 'PASOPIA'
1170 '*
1180 COLOR 7,0
1190 LOCATE 10,0:PRINT" ___                          ●
1200 LOCATE 10,1:PRINT" |   \   _    _    _    __    _
1210 LOCATE 10,2:PRINT" |___/   _\  <_\  / \  |  \  |  _\
1220 LOCATE 10,3:PRINT" |      / V    \ |   | |__/  | / V
1230 LOCATE 10,4:PRINT" |      \__/\ \__/ \__/ |     | \__/\      ";
1240 RETURN
1250 '*
1260 '* POSITION FOR COLOR CHAR
1270 '*
1280 DIM XPOS(10),YPOS(10),CHARDATA$(10,7)
1290 RESTORE 1410
1300 FOR LOOP1=1 TO 10                         ' FOR CIRCLE NUMBER
1310     READ XPOS(LOOP1),YPOS(LOOP1),W1
1320     FOR LOOP2=1 TO W1                        ' FOR DATAN
```

```
1330      READ W2$
1340      FOR LOOP3=0 TO 7                        ' FOR COLOR CHAR
1350        IF W2$="X" THEN CHARDATA$(LOOP1,LOOP3)=CHARDATA$(LOOP1,LOOP3)+ COL$(L
OOP3):GOTO 1370
1360        CHARDATA$(LOOP1,LOOP3)=CHARDATA$(LOOP1,LOOP3)+CHR$(VAL("&H"+W2$))
1370      NEXT LOOP3
1380    NEXT LOOP2
1390 NEXT LOOP1
1400 RETURN
1410 'X DATA FOR COLOR CHAR.
1420 '       XPOS-YPOS-DATAN-DATA----
1430 DATA    40 ,   5,    7,X,1F,1D,X,1F,1D,X
1440 DATA    47 ,   7,   11,X,1F,1D,1D,X,1F,1D,X,1F,1D,X
1450 DATA    49 ,  11,    7,X,1F,1D,X,1F,1D,X
1460 DATA    46 ,  14,    9,X,1F,1D,X,1F,1D,X,1F,X
1470 DATA    40 ,  17,    7,X,1F,1D,X,1F,1D,X
1480 DATA    32 ,  17,    7,X,1F,1D,X,1F,1D,X
1490 DATA    26 ,  14,   10,X,1F,1D,X,1F,1D,X,1F,1D,X
1500 DATA    24 ,  11,    7,X,1F,1D,X,1F,1D,X
1510 DATA     6 ,   7,   10,X,1F,1D,X,1F,1D,X,1F,1D,X
1520 DATA    32 ,   5,    7,X,1F,1D,X,1F,1D,X
```

このプログラムでは，背景色と同じ色で円を描き，その前のアトリビュート・キャラクタを書き換えるだけで色を変化させ，まるで円が動くアニメ(不連続ですが)のように見せています．(カラーページ②参照)

各サブルーチンは，次のようなことを実行しています．

770行～	円の前のアトリビュート・キャラクタを書き換える
850行～	円を描き，PAINTする
1030行～	各行の基本色を背景色と同じにする(LOCATE命令を実行すると，左端に実行時の色のアトリビュート・キャラクタが書き込まれる)，さらに，円の直前に背景色と同じ色を表すアトリビュート・キャラクタを書き込む
1250行～	DATA文より，アトリビュート・キャラクタを書き込む場所と，カーソル移動コードを読み，書き換えるための文字変数(CHARDATA$(円$N_o$，色))を作成する．

1070行でLOCATEだけで行の色を変化させていますが，LOCATEを実行したときに，左端の桁の左側にその直前のCOLORで宣言された色コードを書き込むために色が変化するのです．

3-3　GET@とPUT@

3-3-1　GET@のデータ形式

T-DISKBASICでは，画面のグラフィックを配列に読みとるGET@と配列を画面に描くPUT@を使うことができます．

GET@のために必要な配列のサイズはSCREEN1としSCREEN2で異なり，次のようになっています．

SCREEN1	4＋n／2バイト
SCREEN2	4＋n／8バイト

GET@で読み込まれた画面情報は，次の形式で配列変数に書き込まれます．

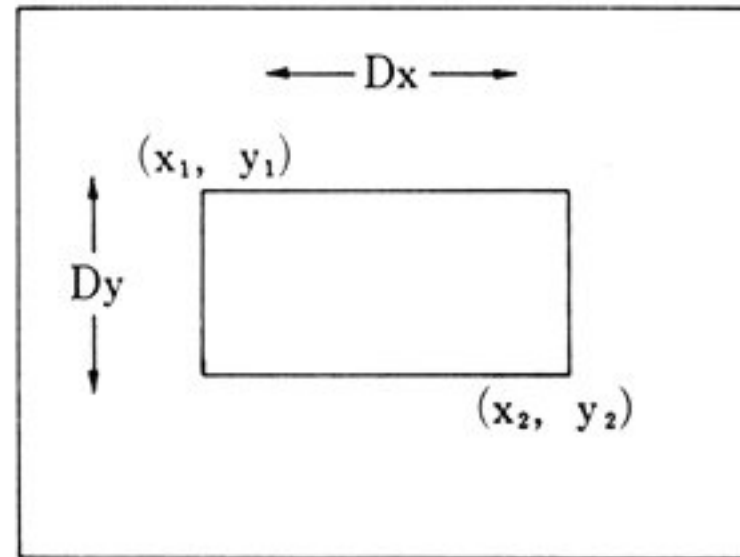
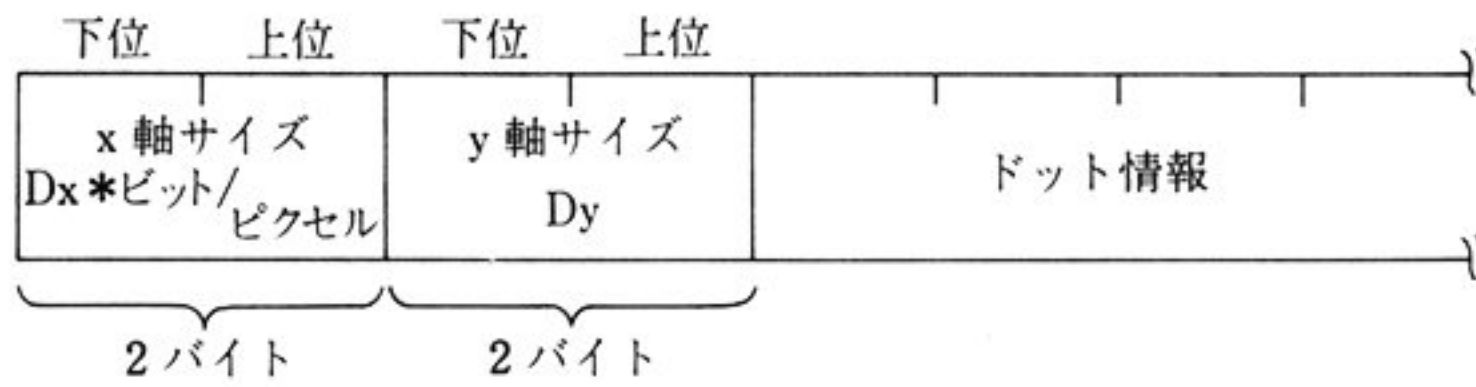
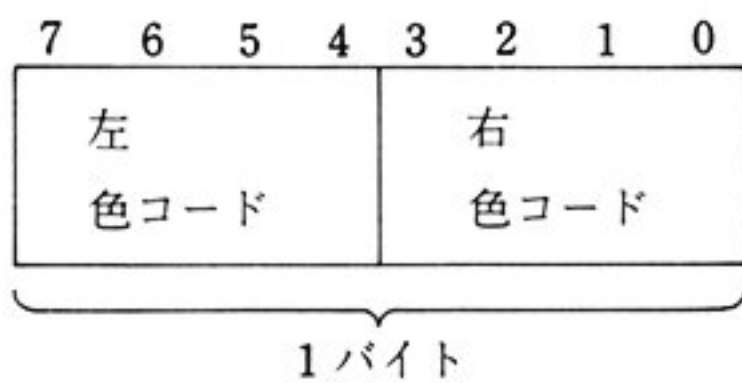
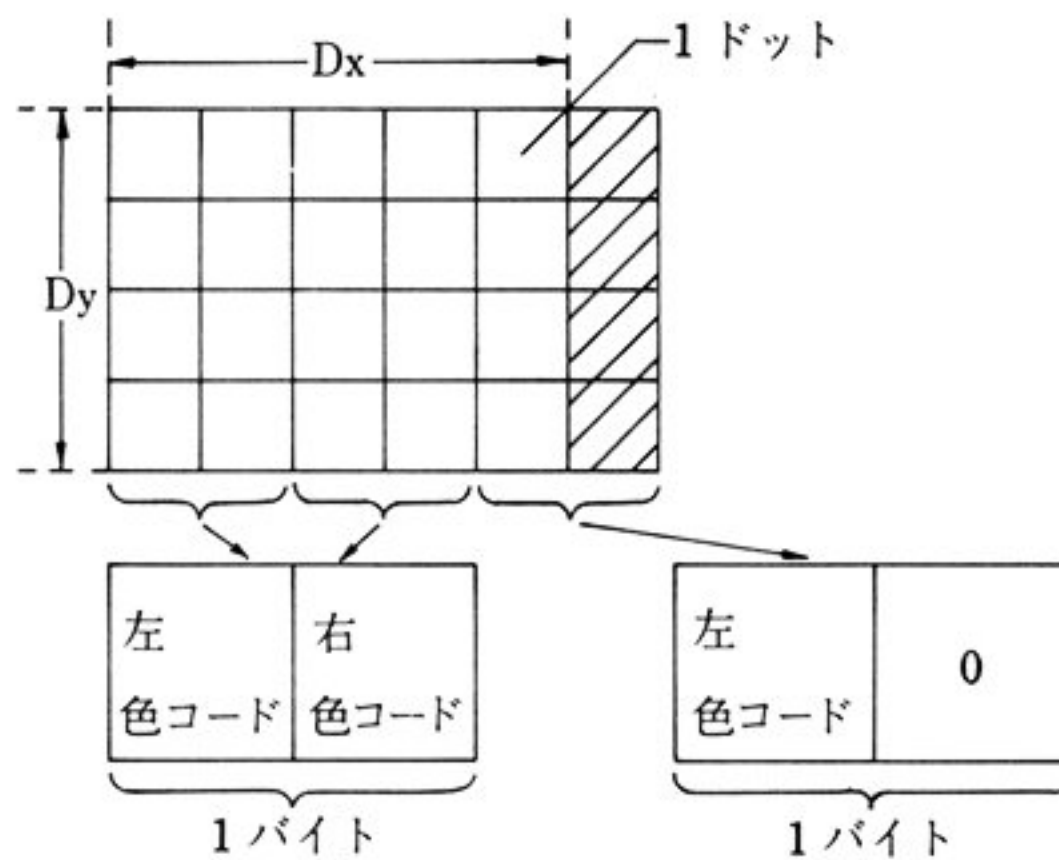
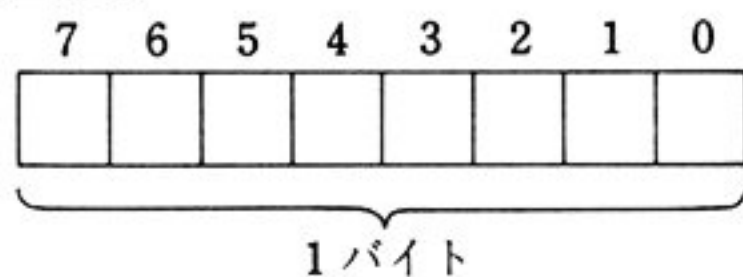
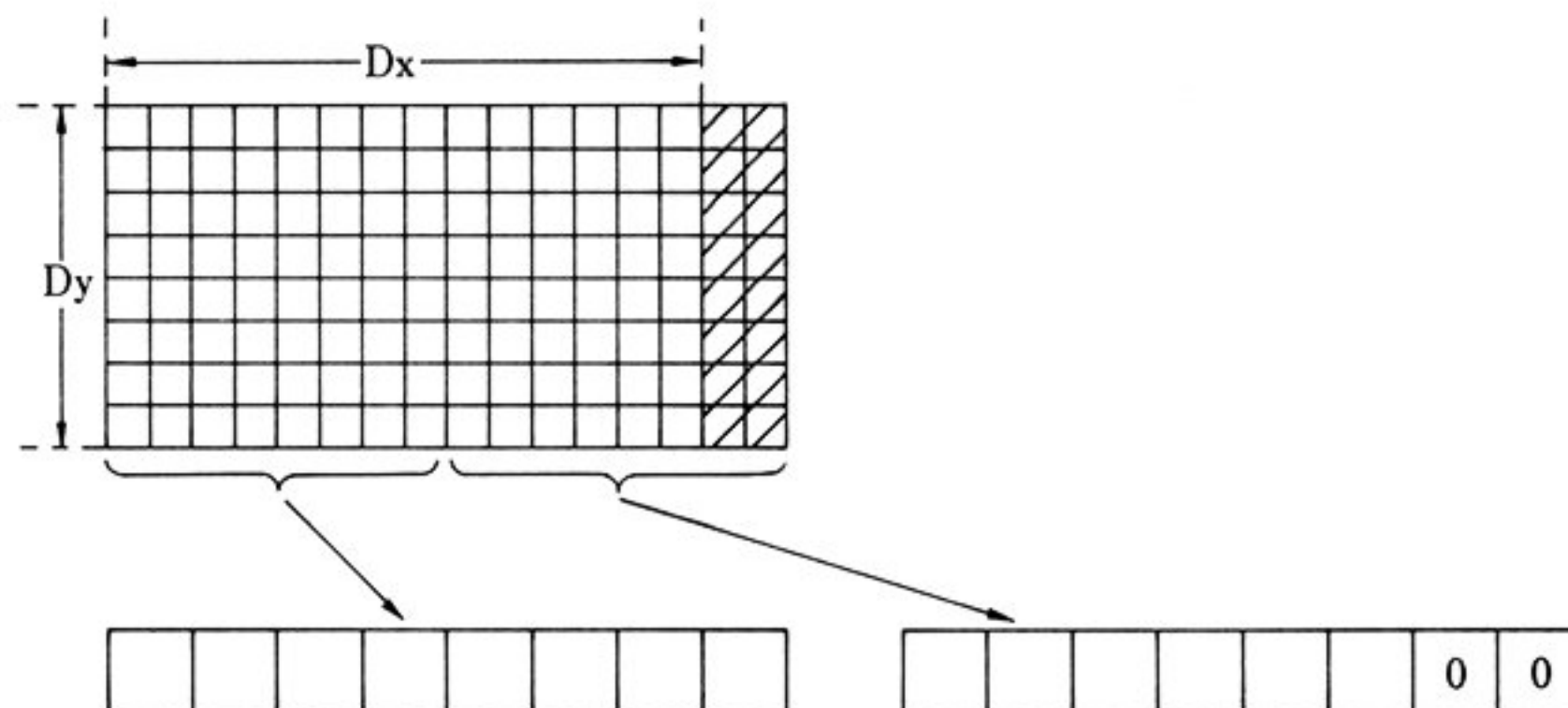



図3－3－1　画面情報の形式

座標(x_1, y_1)を基点として横方向(x軸) 1行のドット情報を8ビットに区切って書き込み，続いて次の行のドット情報を書き込むことを繰り返して(Dx，Dy)の画面情報を格納します．

図3－3－2　格納のされ方

1バイトで2ドットの色コードを表わす．Dxが奇数の場合は横1列の最後のドットを表わすバイトの下位4ビットは0になる．

1バイトで8ドットのON/OFFをビットのON/OFFで表わす．
Dxが8の整数倍でない場合は，横1列の最後のドットを表わすバイトの下位の残りのビットは0になる．

このうち，x軸サイズは，x軸方向のドット数に，1ドットのドット構成数をかけた値が格納されています．

Dx×ビット/ピクセル

Dx……x軸方向のドット数

ビット/ピクセル……SCREEN1：4

SCREEN2：1

y軸サイズは，y軸方向のドット数が格納されます．

ドット情報は，バイトごとに，SCREEN1では各ドットの色コードが，SCREEN2ではドットのON/OFFが格納されています．

3-3-2　ROM-BASICでGET@，PUT@を使う

前出のGET@，PUT@はグラフィックを使用する時には非常に便利ですが，ROM-BASICでは使用できません．そこでROM版でも使用できる高速GET@・PUT@の機械語サブルーチンを紹介します．（カラーページ③参照）

```
100 '***
110 '***      High Speed GET@/PUT@ ( T-BASIC )
120 '***             M-WORD SUB F000H TO F14EH
130 '***
140 '*********  Paramater address  ************************************
150 '*    address                mean                                     *
160 '*   F004H            GET/PUT Buffer address ( LOW  )                 *
170 '*   F005H                                    ( HIGH )                *
180 '*   F006H            GET/PUT ｼﾃｲ   ( 0... GET   1... PUT )           *
190 '*   F008H            GET/PUT ﾉ X ｻﾞﾋｮｳ ( ﾃｷｽﾄ ﾉ X ｻﾞﾋｮｳ ﾃﾞ ｼﾃｲ ｽﾙ｡)  *
200 '*   F009H                        Y                                   *
210 '*   F00AH                        X ﾎｳｺｳ ﾓｼﾞｽｳ                         *
220 '*   F00BH                        Y                                   *
230 '*****************************************************************
240 CLEAR ,&HEFFF
250 WIDTH 80
260 SCREEN 1:CLS
270 GOSUB 530
280 GOSUB 900
320 '*
330 '*    SET PARAMETER AND GET
340 '*
350 POKE &HF004,&H0       :' SET BUFFER ADDRESS LOW
360 POKE &HF005,&HF2      :' SET BUFFER ADDRESS HIGH
370 POKE &HF006,0         :' SET FUNCTION FOR GET@
380 POKE &HF008,3         :' SET X START
390 POKE &HF009,2         :' SET Y START
400 POKE &HF00A,23        :' SET X WIDTH
410 POKE &HF00B,6         :' SET Y WIDTH
420 GETPUT=&HF055
430 CALL GETPUT
440 '*
450 '*    PUT@ START
460 '*
470 POKE &HF006,1         :' SET FUNCTION FOR PUT@
480 FOR I=1 TO 5
490  POKE &HF008,I*3+10 :POKE &HF009,I*3+3:' SET X/Y START
500  CALL GETPUT
510 NEXT I
520 END
530 '*
540 '*    SET M-WORD DATA
550 '*
560 SAD=&HF000
570 READ A$
580 IF A$="END" THEN RETURN
590 POKE SAD,VAL("&H"+A$)
600 SAD=SAD+1
610 GOTO 570
620 '*
630 '*    GET/PUT M-WORD SUBROUTINE DATA
640 '*
650  DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
660  DATA 00,C5,3E,0C,D3,10,DB,11,67,3E,0D,D3,10,DB,11,6F
670  DATA 01,01,00,09,C1,C9,7C,E6,07,67,C9,00,00,00,00,00
680  DATA 00,00,E5,21,00,00,48,06,00,FE,00,CA,42,F0,09,3D
690  DATA 18,F7,E5,C1,E1,C9,DB,09,DB,09,E6,40,20,FA,DB,09
700  DATA E6,40,28,FA,C9,3A,0B,F0,32,0F,F0,ED,5B,04,F0,CD
710  DATA 11,F0,06,51,3A,09,F0,CD,32,F0,09,3A,08,F0,4F,06
720  DATA 00,09,CD,26,F0,22,00,F0,22,02,F0,3A,0A,F0,32,0E
730  DATA F0,3A,0C,F0,32,10,F0,3A,10,F0,FE,00,CA,99,F0,3D
740  DATA 32,10,F0,13,13,13,13,18,EE,3E,04,32,0D,F0,2A,02
750  DATA F0,3A,06,F0,FE,01,CA,32,F1,F3,CD,46,F0,7D,D3,00
760  DATA 7C,F6,40,D3,0A,CD,46,F0,DB,09,47,DB,02,FB,CB,78
770  DATA 20,01,AF,12,13,01,00,10,09,3A,0D,F0,3D,32,0D,F0
780  DATA 20,CF,3A,0E,F0,3D,32,0E,F0,CA,E8,F0,2A,02,F0,23
790  DATA CD,26,F0,22,02,F0,18,9F,3A,0C,F0,FE,00,CA,1D,F1
800  DATA 32,0E,F0,2A,02,F0,23,CD,26,F0,22,02,F0,3E,04,32
810  DATA 0D,F0,3E,00,CD,38,F1,01,00,10,09,3A,0D,F0,3D,32
820  DATA 0D,F0,20,EE,3A,0E,F0,3D,32,0E,F0,20,D6,3A,0F,F0
830  DATA 3D,32,0F,F0,C8,2A,00,F0,01,51,00,09,CD,26,F0,C3
840  DATA 75,F0,1A,CD,38,F1,18,8C,F5,F3,CD,46,F0,F1,D3,01
850  DATA 7D,D3,00,7C,F6,80,D3,0A,CD,46,F0,D3,0A,FB,C9,END
900 '*
```

```
910 '* DRAW TEST DATA
920 '*
930 RESTORE 1000
940 READ X1:IF X1<0 THEN 970
950 READ Y1,X2,Y2,C
960 LINE(X1,Y1)-(X2,Y2),C:GOTO 940
970 READ X1:IF X1<0 THEN RETURN
980 READ Y1,C
990 PSET(X1,Y1),C:GOTO 970
1000 '*
1010 '* TEST DATA
1020 '*
1030 DATA 0,0,49,0,7,0,10,49,10,7,0,0,0,10,7,49,0,49,10,7
1040 DATA 10,10,10,16,1,12,10,14,12,1,14,12,12,14,1,17,10,14,16,1
1050 DATA 17,10,20,16,1,24,10,22,12,1,22,12,25,15,1,24,16,22,16,1
1060 DATA 29,10,26,13,1,26,13,29,16,1,29,16,32,13,1,32,13,29,10,1
1070 DATA 29,11,27,13,4,27,13,29,15,4,29,15,31,13,4,31,13,29,11,4
1080 DATA 33,10,33,16,1,35,10,37,12,1,37,12,35,14,1,39,10,39,16,1
1090 DATA 44,10,41,16,1,44,10,47,16,1,-1
1100 DATA 11,10,1,11,14,1,25,10,1,29,12,2
1110 DATA 28,13,2,29,14,2,30,13,2,34,10,1
1120 DATA 34,14,1,-1
```

このプログラムは，DISKBASICのGET＠.PUT＠より高速ですが，PUT＠は常にPSETのモードで実行されます.

機械語サブルーチンは，次のようにパラメータを設定してからCALLします.

1) GETなら0，PUTなら1をF006HにPOKEする.
2) GETまたはPUTのバッファのアドレスをF004，F005に書き込む.
3) X座標の左側の座標(文字座標)．F008Hに，Y座標の上側をF009HにPOKEする.
4) X方向の文字数をF00AHに，Y方向の文字数をF00BHにPOKEする.
5) USRかCALLで機械語サブルーチンを呼ぶ.

3-4 LINE・PSET・PAINT

T-BASICでは，グラフィックの解像度はSCREEN1で160×100(80文字)，SCREEN2で640×200になります．グラフィックの関係でよく使われる命令としては，LINE，PSET，PAINTがあります．これらの命令には相対座標を使うことができますが，初期に発行されたマニュアルには書かれていません．相対座標を使うときは次のように，座標の前にSTEPをつけます.

例) PSET STEP(10, 10)

次に相対座標を使ったサンプル・プログラムを示します.

```
100 ' ***
110 ' *** LINE SAMPLE
120 ' ***
130 SCREEN 2:WIDTH 80:COLOR 7,0:CLS
140 KEY OFF
150 PI=ATN(1)*4
160 PSET(320,0),0
170 FOR I=45 TO 0 STEP -5
```

```
180    FOR TH=0 TO PI*2 STEP PI/20
190      LINE -STEP(COS(TH)*I,SIN(TH)*10)
200    NEXT
210 NEXT
220 END
```

次に，簡易グラフの作成プログラムを紹介します．このプログラムの使用法は次のとおりです．

1)RUNして，グラフの題名を入力する．

2)折れ線グラフにするか棒グラフにするかを入力．

3)データを入力する(この部分を変えれば，関数を表示することもできます)．

4)目盛の最大値・最小値を入力する．

5)グラフが表示される．もし，プリンタにコピーしたいときはCOPYキーを押す．

プログラムを次に示します．(カラーページ④参照)

```
10 '***
20 '***  ｶﾝｲ ｸﾞﾗﾌ ｻｸｾｲ
30 '***
100 '
110 ' input data
120 '
130 WIDTH 80:SCREEN 2:COLOR 5,0:CLS
140 PRINT TAB(20);"<<<   Graph maker >>>
150 INPUT" Input title of graph";TITLE$
160 PRINT" Input type of graph "
170 PRINT"        ｵﾚｾﾝ.....1  ﾎﾞｳ......2"
180 LOCATE 40,3:PRINT TAB(79);:LOCATE 40,3
190 INPUT A$:PRINT
200 IF A$<>"1" AND A$<>"2" THEN 180
210 TYPE=VAL(A$)
220 INPUT" Input number of data (less than 80)   ";DATAN
230 IF DATAN>80 THEN PRINT CHR$(252);"Number of data too big!":GOTO 220 ELSE IF
DATAN<1 THEN PRINT"Illegal number!":GOTO 220
240 OPTION BASE 1
250 DIM D(DATAN)
260 DATAMAX=-1E+38:DATAMIN=1E+38
270 FOR I=1 TO DATAN
280   PRINT USING"data ###=";I;
290   INPUT D(I)
300   IF D(I)>DATAMAX THEN DATAMAX=D(I)
310   IF D(I)<DATAMIN THEN DATAMIN=D(I)
320   AVE=AVE+D(I)
330 NEXT I
340 INPUT" ﾒﾓﾘ ﾉ ｻｲﾀﾞｲ ｦ ｲﾚﾃ ｸﾀﾞｻｲ";MAX
350 INPUT"         ｻｲｼｮｳ ｦ ｲﾚﾃ ｸﾀﾞｻｲ";MIN
360 IF MAX<=MIN THEN PRINT CHR$(252);"Error MAX<=MIN":GOTO 340
370 '
380 '   Make graph
390 '
400 CLS
410 LOCATE 20,1:PRINT "<< ";TITLE$;" >>"
420 LINE(0,0)-(639,159),,B
430 LINE(25,30)-(30,30)
440 LINE-STEP(0,100)
450 LINE-STEP(420,0)
460 LOCATE 1,30¥8:PRINT MAX
470 LOCATE 1,130¥8:PRINT MIN
480 STE=1/DATAN*400
490 LOCATE (30+STE*3/2)¥8,18:PRINT "1"
500 AVE=AVE/DATAN
510 YAVE=130-(AVE-MIN)/(MAX-MIN)*100
520 IF YAVE<20 THEN YAVE=20 ELSE IF YAVE>130 THEN YAVE=130
530 '
540 FOR I=1 TO DATAN
```

```
550    Y=130-(D(I)-MIN)/(MAX-MIN)*100
560    IF Y<20 THEN Y=20 ELSE IF Y>130 THEN Y=130
570    X=30+I*STE
580    ON TYPE GOSUB 690,790
590 NEXT I
600 IF TYPE=1 THEN LINE(30,YAVE)-STEP(400,0)
610 LOCATE X1¥8-1,18:PRINT DATAN
620 C$=CHR$(254):C1$=CHR$(253)
630 LOCATE 58,10:PRINT C$;" ﾃﾞｰﾀ ｽｳ :";DATAN   ;C1$
640 LOCATE 58,12:PRINT C$;" ﾍｲｷﾝ    :";AVE     ;C1$
650 LOCATE 58,15:PRINT C$;"   Max   :";DATAMAX;C1$
660 LOCATE 58,16:PRINT C$;"   Min   :";DATAMIN;C1$
670 LOCATE 0,20
680 END
690 '
700 ' Oresen sub
710 '
720 IF I=1 THEN 740
730 LINE(X1,Y1)-(X,Y)
740 X1=X:Y1=Y
750 LINE(X,130)-STEP(0,5)
760 IF I MOD 5=0 THEN LINE-STEP(0,2)
770 RETURN
780 '
790 ' Bou sub
800 '
810 LINE(X,130)-(X+STE-2,Y),,B
820 IF Y>YAVE THEN 840
830 LINE-(X,YAVE),,BF
840 X1=X+STE/2+1
850 LINE(X1,130)-STEP(0,5)
860 IF I MOD 5=0 THEN LINE-STEP(0,2)
870 RETURN
```

3-5　グラフィックテクニック

3-5-1　スクリーンモード1.5(160×200フルカラーモード)

パソピアは，マニュアルには書かれていませんが，160×200ドットで8色使うことが可能です．次のプログラムを実行してみて下さい．

```
1000 '***
1010 '***          SCREEN DOT 160*200
1020 '***
1030 WIDTH 80
1040 CLS
1050 SCREEN 2
1060 OUT 8,224     ': SELECT SCREEN MODE 160*200
1070 X=0:Y=0:C=1:D=1:E=1
1080 GOSUB 1190     ' PSET
1090 '
1100 IF X>158 THEN D=-1:C=(C+1) MOD 7+1
1110 IF X<1   THEN D=1 :C=(C+1) MOD 7+1
1120 IF Y>198 THEN E=-1:C=(C+1) MOD 7+1
1130 IF Y<1   THEN E=1 :C=(C+1) MOD 7+1
1140 X=X+D:Y=Y+E
1150 GOTO 1080
1160 '*
1170 '* PSET SUB
1180 '*
1190 B1=C MOD 2:B2=(C¥2) MOD 2:B3=(C¥4) MOD 2
1200 X1=X*4+1
1210 PSET(X1,Y),B1      :' GREEN
1220 PSET(X1+1,Y),B2    :' RED
1230 PSET(X1+2,Y),B3    :' BLUE
1240 RETURN
```

このプログラムを実行すると，色分解能160×200で描線することができます．オールBASICであるため，実行速度は遅いのですが，ハード的には160×200の色指定をすることが可能です．

140行のOUT8, 224でこのモードが設定されます．スクリーンモードと8番ポートは次のような関係になっています．

スクリーンモード	横方向文字数（ドット数）	8番ポートに出力する値							
		背景色0	背景色1	背景色2	3	4	5	6	7
0	36（ —— ）	0	1	2	3	4	5	6	7
	80（ —— ）	32	33	34	35	36	37	38	39
1	36（ 72×96 ）	64	65	66	67	68	69	70	71
	80（160×100）	96	97	98	99	100	101	102	103
2	36（288×192）	128	129	130	131	132	133	134	135
	80（640×200）	160	161	162	163	164	165	166	167
1.5（？）	36（ 72×192）	192	193	194	195	196	197	198	199
	80（160×200）	224	225	226	227	228	229	230	231

図3－5－1　各スクリーンモードとモード1.5の8番ポートへの出力の値

8番ポートはCRTCの8255にスクリーンモードと背景色を出力するポートで，各ビットは次のような意味を持ちます．

BIT7　ファイン・グラフィックモードセット信号

DIT6　グラフィックモードセット信号

BIT5　WIDTHセット（80文字… 1）

BIT4～3未使用

BIT2—0背景色セット信号

（液晶ディスプレイでは1：10ラスタ　0：8ラスタ）

これからもわかるように，このモードではファイングラフィックモードとグラフィックモードの両方がセットされた形になるので横方向160ドットで8色の色指定ができ，さらに縦方向の分解能が200ドットというスクリーンモードになります．このときのディスプレイコードはSCREEN 1のときと同じになるので，SCREEN2でPSETを行うと横方向の4ドット分で1ドットとして表示されるのです．

このとき各ドットは次の色を示します．

xをドットのx座標とすると，

x　MOD　4＝0のドット：無視される

x　MOD　4＝1のドット：緑のピクセル（画素）

x　MOD　4＝2のドット：赤のピクセル

x　MOD　4＝3のドット：青のピクセル

画面に表示される色は，緑，赤，青の3つのピクセルの和となります．

このモードを使えば，タイリング（Tiling：となりあわせた色をまぜて中間色を表す）してもなか

なか美しく見えます．タイリングのサンプルプログラムを次に示します．中解像モニタの方が高解像カラーモニタよりきれいに色が混ってみえるようです．（カラーページ⑤参照）

```
100 '***
110 '***  TILING T-BASIC
120 '***
130 '
140 WIDTH 80
150 CLS
160 SCREEN 2                    :' SET SCREEN MODE
170 OUT 8,224                   :' FOR CRTC
180 DIM C(1)
190 FOR Y1=0 TO 160 STEP 21
200    FOR XX=0 TO 159 STEP 20
210       C(1)=C(1)-1:IF C(1)<0 THEN C(1)=7:C(0)=C(0)-1:IF C(0)<0 THEN C(0)=7
220          FOR I=0 TO  17:X=XX+I
230            FOR J=0 TO 19 :C=C((I+J) MOD 2):Y=Y1+J
240               GOSUB 330
250            NEXT J
260          NEXT I
270       NEXT XX
280 NEXT Y1  :LOCATE 0,23
290 END
300 '*
310 '* PSET ROUTINE
320 '*
330 C0=C MOD 2:C1=(C¥2) MOD 2:C2=(C¥4) MOD 2
340 X1=X*4+1
350 PSET(X1,Y),C2
360 PSET(X1+1,Y),C1
370 PSET(X1+2,Y),C0
380 RETURN
```

3-5-2　カラーREMARK

ゲームソフトなどで，REM文が着色されているものを見たことがあると思います．T-BASICでは，難しいテクニックなど必要とせずにリストに着色することができます．その方法は，ファンクションキーにカラーアトリビュート・キャラクタを入れ，適当なところで，スペースの代わりに押せばよいのです．（カラーページ⑥参照）

KEY　1，CHR$(254)

でPF1に黄色のキャラクタが入りました．REMや"'"のあとに書き込めば注釈文に色が付きます．

3-5-3　VIEW・WINDOW

某メーカーの某機種には仮想スクリーンを扱う命令として，VIEW命令とWINDOW命令が付いています．パソピアには付いていないので悔しい思いをした人がいるのではないかと思いますが，無いものは作ってしまえばよいのです．BASICで組んでも，パソピアはかなり高速なので十分実用になると思います．（カラーページ⑦参照）

```
10 '***
20 '***         VIEW & WINDOW SAMPLE
30 '***
100 WIDTH 80:SCREEN 2:COLOR 5,7:CLS
110 GOSUB 10000           :'  INIT VIEW & WINDOW
120 '
130 ' WINDOW SET
140 '
150 WINDOWX1=-10:WINDOWX2=160
160 WINDOWY1=  0:WINDOWY2=70
170 '
180 ' VIEW DATA SET FOR ARRAY
190 '
200 LOOPMAX=3
210 OPTION BASE 1
220 DIM VX(2,LOOPMAX),VY(2,LOOPMAX),COL(LOOPMAX)
230 RESTORE
240 FOR I=1 TO LOOPMAX
250    FOR J=1 TO 2
260 READ VX(J,I),VY(J,I)
270 NEXT J,I
280 DATA 0,0,624,192
290 DATA 72,56,480,132
300 DATA 108,75,340,124
310 COL(1)=1:COL(2)=2:COL(3)=6 :COLP=0
320 '
330 ' MAIN START
340 '
350 FOR LOOP=1 TO LOOPMAX
360 VIEWX1=VX(1,LOOP):VIEWX2=VX(2,LOOP)
370 VIEWY1=VY(1,LOOP):VIEWY2=VY(2,LOOP)
380 RESTORE 1030
390 GOSUB 10130           :' CLS VIEW-PORT
400 COL=COL(LOOP)
410 GOSUB 690
420 COLP=COL
430 READ A$
440 IF A$="P" THEN GOSUB 620    :' PAINT
450 IF A$="0" THEN GOSUB 500    :' LINE
460 IF A$="END" THEN 480
470 GOTO 430
480 NEXT LOOP
490 END
500 '
510 ' LINE
520 '
530 READ X1 ,Y1,X2,Y2
540 X=FNVWX(X1):Y=FNVWY(Y1)
550 XX=FNVWX(X2):YY=FNVWY(Y2)
560 LINE(X,Y)-(XX,YY)
570 READ X:IF X<0 THEN RETURN
580 READ Y
590 XX=FNVWX(X):YY=FNVWY(Y)
600 LINE-(XX,YY)
610 GOTO 570
620 '
630 ' PAINT
640 '
650 READ X,Y,CC,CD
660 XX=FNVWX(X):YY=FNVWY(Y)
670 PAINT(XX,YY),1,1
680 RETURN
690 '
700 ' COLOR FOR VIEW-PORT
710 '
720 XS=VIEWX1¥8+1:XE=VIEWX2¥8-1
730 YS=VIEWY1¥8+1:YE=VIEWY2¥8-1
740 FOR I=YS TO YE
750   LOCATE XS,I:PRINT CHR$(248+COL)
760   LOCATE XE,I:PRINT CHR$(248+COLP)
770 NEXT I
780 RETURN
1000 '
```

```
1010 ' DATA FOR PICTURE
1020 '
1030 DATA 0   ,29  ,10  ,3   ,55  ,15  ,55  ,10  ,49  ,29
1040 DATA 49  ,29  ,55  ,40  ,55  ,40  ,10  ,29  ,10  ,-1
1050 DATA 0   ,29  ,31  ,21  ,43  ,29  ,43  ,29  ,31  ,-1
1060 DATA P   ,35  ,13  ,5   ,5   ,0   ,45  ,10  ,44  ,10
1070 DATA 43  ,12  ,42  ,14  ,42  ,23  ,42  ,27  ,62  ,42
1080 DATA 63  ,43  ,63  ,45  ,62  ,47  ,55  ,47  ,53  ,46
1090 DATA 52  ,45  ,52  ,43  ,42  ,43  ,42  ,48  ,43  ,52
1100 DATA 47  ,54  ,52  ,55  ,65  ,55  ,70  ,54  ,72  ,53
1110 DATA 73  ,52  ,74  ,48  ,74  ,42  ,73  ,39  ,53  ,23
1120 DATA 52  ,22  ,53  ,10  ,54  ,17  ,55  ,17  ,62  ,17
1130 DATA 64  ,20  ,64  ,23  ,75  ,23  ,75  ,15  ,74  ,13
1140 DATA 73  ,12  ,71  ,11  ,70  ,10  ,45  ,10  ,-1  ,P
1150 DATA 45  ,13  ,1   ,1   ,0   ,81  ,10  ,78  ,13  ,77
1160 DATA 15  ,77  ,47  ,78  ,50  ,79  ,53  ,82  ,54  ,84
1170 DATA 55  ,85  ,55  ,102 ,55  ,104 ,54  ,105 ,53  ,108
1180 DATA 51  ,109 ,50  ,109 ,43  ,98  ,43  ,98  ,47  ,97
1190 DATA 48  ,89  ,48  ,87  ,46  ,87  ,20  ,88  ,17  ,90
1200 DATA 16  ,96  ,16  ,97  ,17  ,98  ,18  ,98  ,23  ,109
1210 DATA 23  ,109 ,15  ,107 ,13  ,105 ,11  ,104 ,10  ,81
1220 DATA 10  ,-1  ,P   ,90  ,13  ,1   ,1   ,0   ,111 ,10
1230 DATA 111 ,55  ,123 ,55  ,123 ,10  ,111 ,10  ,-1  ,P
1240 DATA 120 ,13  ,1   ,1   ,0   ,127 ,10  ,127 ,55  ,130
1250 DATA 55  ,130 ,10  ,127 ,10  ,-1  ,P   ,130 ,13  ,1
1260 DATA 1   ,END
10000 '***
10010 '***  VIEW & WINDOW SUBROUTINE
10020 '
10030 '
10040 '     VIEW/WINDOW INITILISE
10050 '
10060 VIEWX1=0:VIEWX2=639         :' WIDTH 80:SCREEN 2 / ﾄｷ
10070 VIEWY1=0:VIEWY2=199
10080 WINDOWX1=0:WINDOWX2=639
10090 WINDOWY1=0:WINDOWY2=199
10100 DEF FNVWX(X)=(VIEWX2-VIEWX1)/(WINDOWX2-WINDOWX1)*(X-WINDOWX1)+VIEWX1
10110 DEF FNVWY(Y)=(VIEWY2-VIEWY1)/(WINDOWY2-WINDOWY1)*(Y-WINDOWY1)+VIEWY1
10120 RETURN
10130 '
10140 '   CLEAR VIEW-PORT
10150 LINE(VIEWX1,VIEWY1)-(VIEWX2,VIEWY2),0,BF
10160 LINE(VIEWX1,VIEWY1)-(VIEWX2,VIEWY2),1,B
10170 RETURN
```

このプログラムを実行すると，ASCIIのロゴタイプが大きさを変えて表示します．

このプログラムでは，10000行からのサブルーチンでVIEW・WINDOWの初期設定を行い，10140からのサブルーチンでVIEWポートの画面クリアを行います．VIEWとWINDOWは次の図のようになります．

本来ならばVIEWポートの外側にはみ出す部分ははみ出さないように補正するべきなのですが，このプログラムでは補正していませんので注意して下さい．

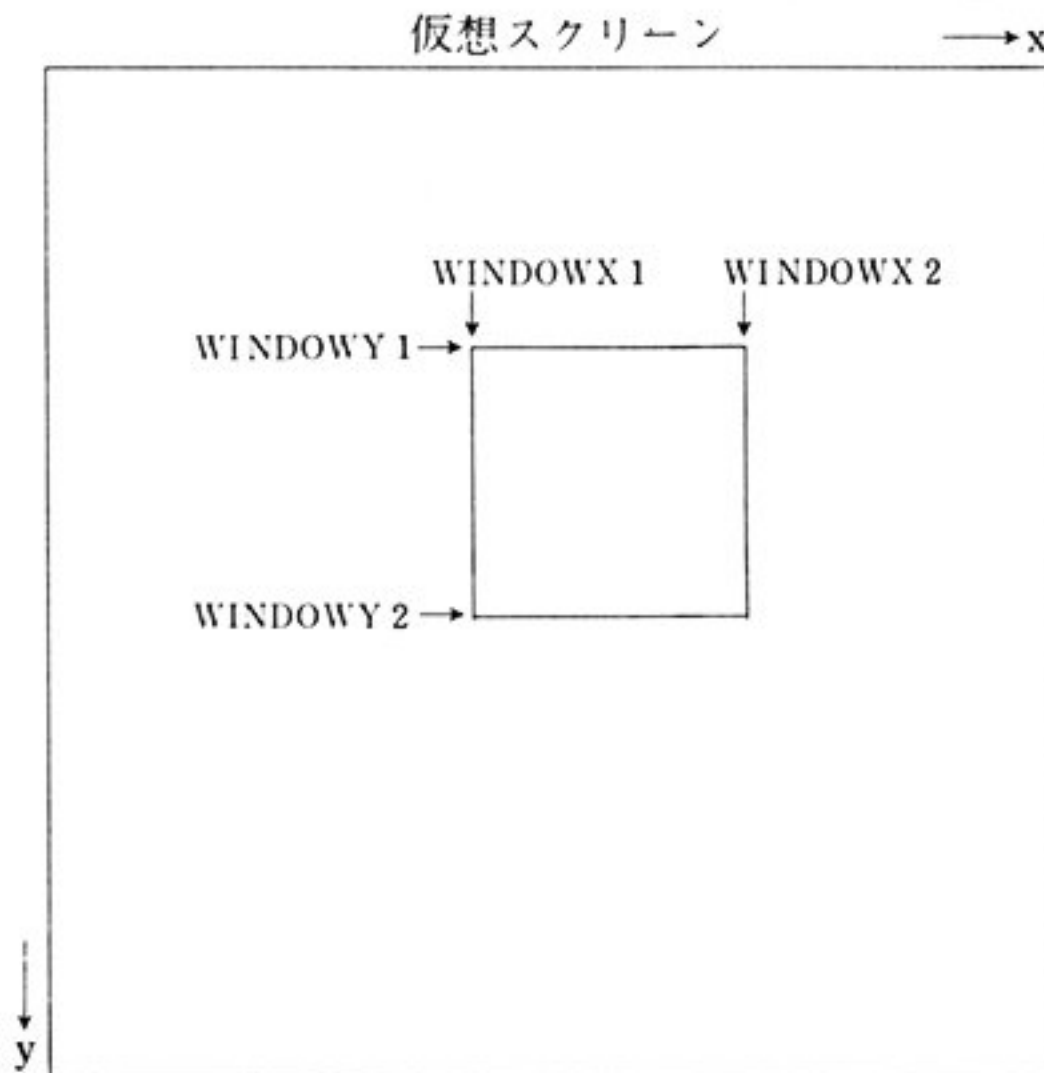
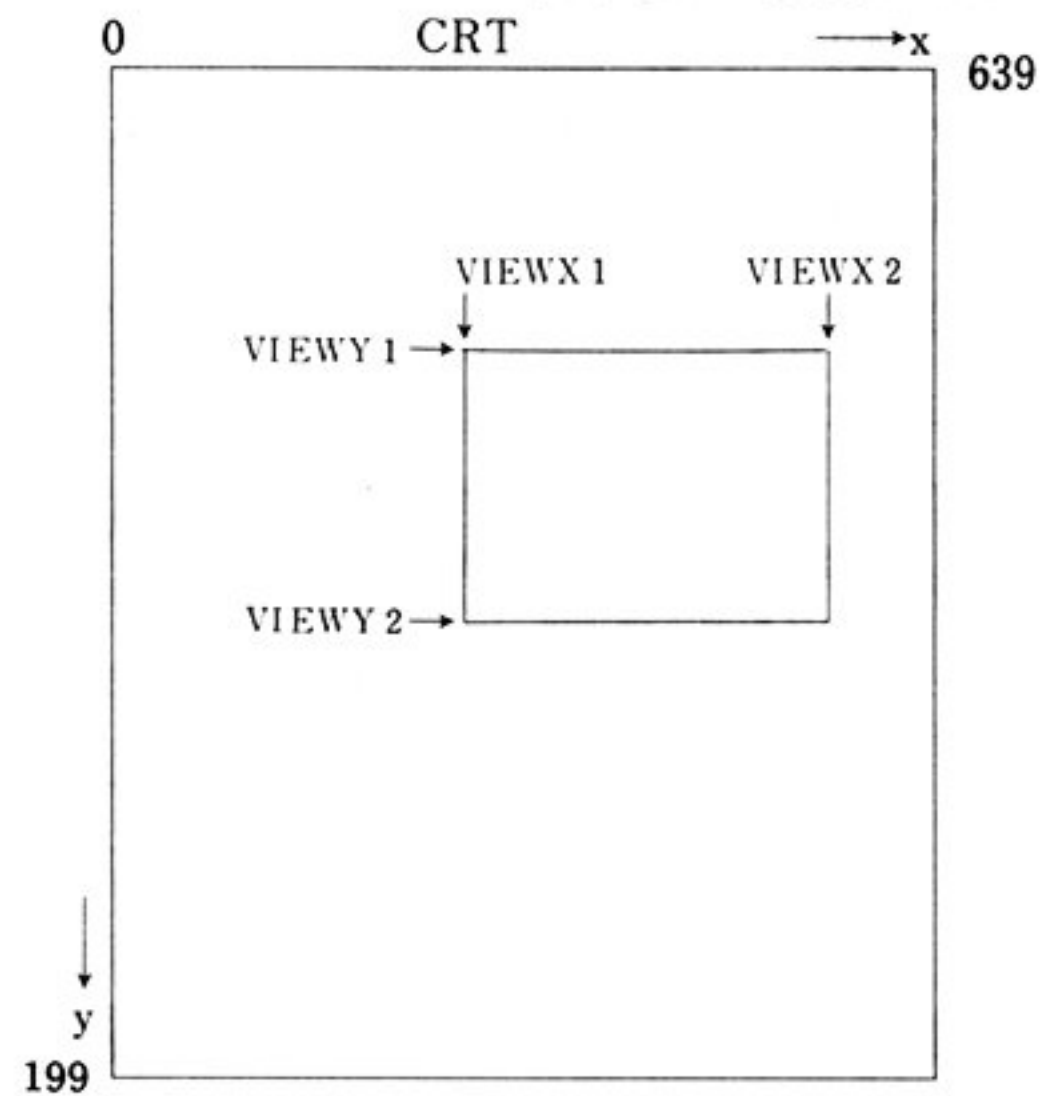



図3－5－2　WINDOWとVIEW

3-5-4　超高速グラフィック

パソピアのグラフィック機能の特色を生かした高速グラフィックのサンプルを紹介します．次のプログラムでは色の変化と画面の変化を高速に行います．（カラーページ⑧⑨⑩参照）

```
10 '**
20 '**   SCREEN 2 COLOR CHANGE SAMPLE PROGRAM
30 '**
40 '**               ｶﾗｰﾓﾆﾀ ﾃﾞ ｺﾞﾗﾝｸﾀﾞｻｲ
50 '**
100 SCREEN 2:WIDTH 80:DEFINT A-Z
110 COLOR 0,0:CLS
120 LOCATE 30,13:COLOR 7:PRINT"COLOR DEMONSTRATION"
130 '***
140 '*** MAKE CRT READY
150 '***
160 FOR I=0 TO 320 STEP  10
170   LINE(I,    0)-(160+I/2, 87)
180   LINE(640-I,0)-(480-I/2, 87)
190   LINE(I,  199)-(160+I/2,120)
200   LINE(640-I,199)-(640-160-I/2,120)
210 NEXT
220 '***
230 '*** CHANGE COLOR
240 '***
250 LOCATE 30,13:PRINT"CHANGE COLOR
260 FOR II=0 TO 5
270   FOR L=7 TO 0 STEP -1
280     COLOR L
290       FOR J=10 TO 0 STEP -1
300         LOCATE 0,J:LOCATE 0,24-J
310       NEXT
320     NEXT
330   NEXT
340 COLOR 7
350 '***
360 '*** CHANGE SCREEN MODE
370 '***
380 LOCATE 30,13:PRINT"CHANGE SCREEN MODE";TAB(50)
```

```
390 OUT 8,224
400 FOR I=1 TO 2000:NEXT
410 OUT 8,96
420 FOR I=1 TO 2000:NEXT
430 COLOR 7,0:FOR I=0 TO 24:LOCATE 0,I:NEXT
440 FOR I=1 TO 1000:OUT 8,224 :OUT 8,160   :                          NEXT I
450 GOTO 250
```

このプログラムでは，カラー(表示色)を変えることをCOLORとLOCATEだけで行っているためこれだけ高速に色の変更ができるのです．また，スクリーンモードをOUT命令で変化させているので，画面を消さずにモアレの模様を出すことができます．

プログラムの各部分は次のようなことをしています．

100～110行	画面の初期設定．COLOR 0，0 でクリアするのは，表示色を黒として，引いた線が見えないようにするため．
160～210行	画面には見えないが，黒で線を引いている．以後この線を消すことはない．
240～330行	COLORで表示色を設定し，LOCATEでカーソルを動かすことにより，画面左端の左側のアトリビュートに色コードを書き込む．この方法により色の高速変化が可能．
390行	OUT　8,224で160×200ドットモードにしている．
410行	同様に，160×100ドットモードにする．
440行	640×200モードと 8 色160×200モードを急速に変化させる．スペースの数によって画面の様子が変化する．

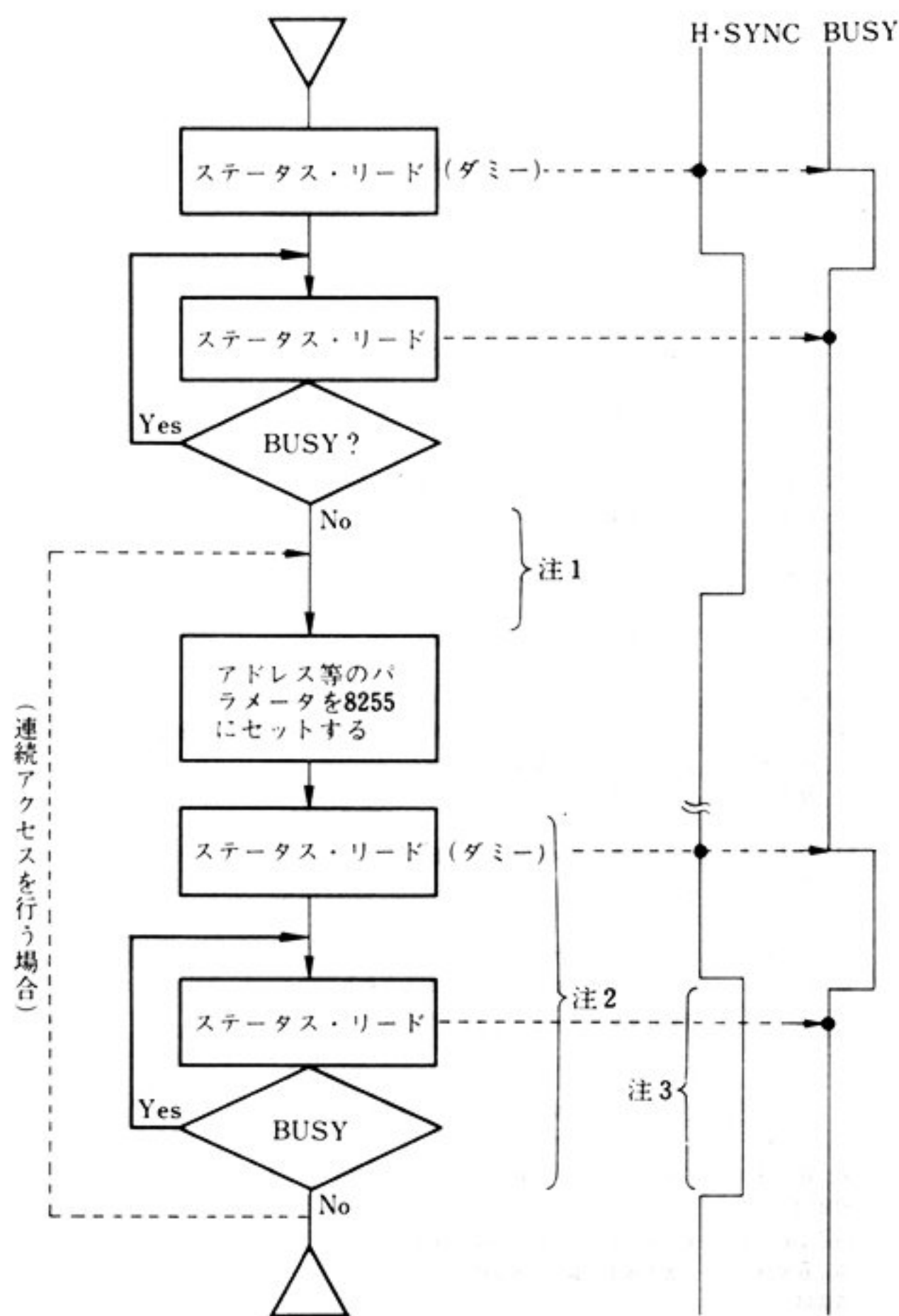
3-5-5　機械語によるV-RAMのRead／Write

機械語を使ってV-RAMを直接アクセスするときには次のフローチャートに示す手順に従う必要があります．

次のプログラムはこの手順に従ってV-RAMのアクセスを行っています．機械語プログラムのデータの部分にそれぞれの命令の意味をREM文で記してあります．

```
1000 '***
1010 '***  V-RAM READ WRITE SAMPLE
1020 '***          T-BASIC
1030 DIM A1(2),A2(2),A3(2)
1040 RESTORE
1050 VRWR=&HF000:IF VRWR<0 THEN VRWR=VRWR+2^16:' V-RAM WRITE ROUTINE
1060 VRRD=&HF100:IF VRRD<0 THEN VRRD=VRRD+2^16:' V-RAM READ  ROUTINE
1070 MADR=VRWR:GOSUB 1530                       :' SET   WRITE ROUTINE
1080 MADR=VRRD:GOSUB 1530                       :' SET   READ  ROUTINE
1090 VRRDA=VRRD+47:VRRDB=VRRD+48
1100 POKE VRRD+38,VRRDA-INT(VRRDA/256)*256
1110 POKE VRRD+39,INT(VRRDA/256)
1120 POKE VRRD+43,VRRDB-INT(VRRDB/256)*256
1130 POKE VRRD+44,INT(VRRDB/256)
1140 '
1150 '   TEST DATA DISPLAY
1160 '
1170 RESTORE 2110:WIDTH 36:SCREEN 1:CLS  :' GRAPHIC DISPLAY MODE
1180 READ X1:IF X1<0 THEN 1210
1190 READ Y1,X2,Y2,C                     :' READ PARAMETER
1200 LINE(X1,Y1)-(X2,Y2),C:GOTO 1180
1210 READ X1:IF X1<0 THEN 1240
```

```
1220 READ Y1,C                              :' READ PARAMETER
1230 PSET(X1,Y1),C:GOTO 1210
1240 '
1250 ' COPY TEST DATA
1260 '
1270 OUT &H10,12:STA=INP(&H11)*256
1280 OUT &H10,13:STA=STA+INP(&H11)+1      :' V-RAM START ADDRESS
1290 IF STA>2047 THEN STA=STA-2048:GOTO 1290:' START ADDRESS MASKING 2KB
1300 FOR ROW=2 TO 4
1310   A1(1)=STA+ROW*37+4
1320   FOR CPY=1 TO 3
1330     A1(2)=STA+(ROW+CPY*4)*37+4+CPY*3
1340     FOR BLK=0 TO 12288 STEP 4096    :' WRITE DATA THREE TIMES
1350       FOR CLM=0 TO 21
1360         FOR I=1 TO 2
1370           A2(I)=A1(I)+CLM
1380           IF A2(I)>2047 THEN A2(I)=A2(I)-2048
1390           A3(I)=A2(I)+BLK
1400         NEXT I
1410         POKE VRRD+16,A3(1)-INT(A3(1)/256)*256:' V-RAM ADDRESS
1420         POKE VRRD+20,INT(A3(1)/256)
1430         CALL VRRD                          :'  READ TEST DATA
1440         OUT &H1,PEEK(VRRDB)                :'  READ DATA TO WRITE DATA
1450         POKE VRWR+16,A3(2)-INT(A3(2)/256)*256:' V-RAM ADDRESS
1460         POKE VRWR+20,INT(A3(2)/256)+PEEK(VRRDA)
1470         CALL VRWR                          :'  WRITE COPY DATA
1480       NEXT CLM
1490     NEXT BLK
1500   NEXT CPY
1510 NEXT ROW
1520 END
1530 '
1540 ' MACHINE COMMAND SET
1550 READ D$:D=VAL("&H"+D$)               :' COMMAND READ
1560 IF D=255 THEN RETURN
1570 POKE MADR,D                          :' COMMAND SET TO MEMORY
1580 MADR=MADR+1:GOTO 1550
1590 '
1600 ' *** WRITE ROUTINE ***
1610 DATA F3    :'     DI                 ;disable interrupt
1620 DATA DB,09 :'     IN     A,09H       ;dummy read
1630 DATA DB,09 :'     IN     A,09H       ;status read
1640 DATA E6,40 :'     AND    40H
1650 DATA 20,FA :'     JR     NZ,*-4      ;wait H-SYNC ON
1660 DATA DB,09 :'     IN     A,09H       ;status read
1670 DATA E6,40 :'     AND    40H
1680 DATA 28,FA :'     JR     Z,-4        ;wait H-SYNC OFF
1690 DATA 3E,00 :'     LD     A,00H
1700 DATA D3,00 :'     OUT    00H,A       ;V-RAM adrs low
1710 DATA 3E,00 :'     LD     A,00H
1720 DATA E6,BF :'     AND    0BFH        ;write signal = low-level
1730 DATA D3,0A :'     OUT    0AH,A       ;adrs high & write signal set
1740 DATA DB,09 :'     IN     A,09H       ;dummy read
1750 DATA DB,09 :'     IN     A,09H       ;status read
1760 DATA E6,40 :'     AND    40H
1770 DATA 20,FA :'     JR     NZ,*-4      ;wait H=SYNC ON
1780 DATA DB,09 :'     IN     A,09H       ;status read
1790 DATA E6,40 :'     AND    40H
1800 DATA 28,FA :'     JR     Z,*-4       ;wait H-SYNC OFF
1810 DATA D3,0A :'     OUT    0AH,A       ;write signal reset
1820 DATA FB    :'     EI                 ;enable interrupt
1830 DATA C9,FF :'     RET
1840 '  *** READ ROUTINE ***
1850 DATA F3    :'     DI                 ;disable interrupt
1860 DATA DB,09 :'     IN     A,09H       ;dummy read
1870 DATA DB,09 :'     IN     A,09H       ;status read
1880 DATA E6,40 :'     AND    40H
1890 DATA 20,FA :'     JR     NZ,*-4      ;wait H-SYNC ON
1900 DATA DB,09 :'     IN     A,09H       ;status read
1910 DATA E6,40 :'     AND    40H
1920 DATA 28,FA :'     JR     Z,*-4       ;wait H-SYNC OFF
1930 DATA 3E,00 :'     LD     A,00H
1940 DATA D3,00 :'     OUT    00H,A       ;V-RAM adrs low
```

```
1950 DATA 3E,00 :'      LD      A,00H
1960 DATA F6,40 :'      OR      40H         ;read signal = high level
1970 DATA D3,0A :'      OUT     0AH,A       ;adrs high & read signal set
1980 DATA DB,09 :'      IN      A,09H       ;dummy read
1990 DATA DB,09 :'      IN      A,09H       ;status read
2000 DATA E6,40 :'      AND     40H
2010 DATA 20,FA :'      JR      NZ,*-4      ;wait H-SYNC ON
2020 DATA DB,09 :'      IN      A,09H       ;read bit-8
2030 DATA E6,80 :'      AND     80H
2040 DATA 32,0,0:'      LD      (VRRDA),A
2050 DATA DB,02 :'      IN      A,02H       ;read data
2060 DATA 32,0,0:'      LD      (VRRDB),A
2070 DATA FB    :'      EI                  ;enable interrupt
2080 DATA C9,FF :'      RET                 ;return
2090 '
2100 '  *** TEST DATA PARAMATER ***
2110 DATA 8,8,49,8,7,8,18,49,18,7,8,8,8,18,7,49,8,49,18,7
2120 DATA 10,10,10,16,1,12,10,14,12,1,14,12,12,14,1,17,10,14,16,1
2130 DATA 17,10,20,16,1,24,10,22,12,1,22,12,25,15,1,24,16,22,16,1
2140 DATA 29,10,26,13,1,26,13,29,16,1,29,16,32,13,1,32,13,29,10,1
2150 DATA 29,11,27,13,4,27,13,29,15,4,29,15,31,13,4,31,13,29,11,4
2160 DATA 33,10,33,16,1,35,10,37,12,1,37,12,35,14,1,39,10,39,16,1
2170 DATA 44,10,41,16,1,44,10,47,16,1,-1
2180 DATA 11,10,1,11,14,1,25,10,1,29,12,2
2190 DATA 28,13,2,29,14,2,30,13,2,34,10,1
2200 DATA 34,14,1,-1
```

注1：この間，CRTの場合54μs以上，LCDの場合は63μs以上の時間が必要です．これはH-SYNCの期間中にアドレスを変化させることによる誤動作を防止するためです．

注2：読み出し書き込みを確認するためのサイクルです．

注3：この間に読み出し書き込みが行われています．

図3－5－3　V-RAMアクセスの手順

第4章

入出力装置

第4章　入出力装置

T-BASICを対象に，カセット，ディスク等のフォーマットや，ファイルの入出力について解説し，使用の際知っておくと便利なルーチンを載せました．特に，パソピアの特徴であるRAMPACについては，別に項を立て解説します．

4-1　オーディオ・カセット

パソピアのカセットインタフェィスは1600ボー（1秒間に1600ビット）と高速で，走行速度が±10％程度変動しても再生可能な変調方式（F-2F）になっています．次にカセットファイルの解説をしましょう．

4-1-1　CLOAD，CSAVE（トークン・ファイル）

T-BASICでは，CLOAD，CSAVEに使用するファイル名に変数を使うことができます．

```
10 '** Casset interfase test
20 FOR I=33 TO 248
30 PRINT CHR$(I);
40 NEXT
csave"TEST"
OK
A$="TEST":CLOAD A$
Found:TEST
OK
```

CSAVEでは，カセットテープに次の図に示すフォーマットで書き込まれます．

図4—1—1　トークンファイルのロジカル構造

図4—1—2　フィジカルフォーマット

データは256バイトずつに分割して書き込まれています．

最初に書き込まれる10バイトのD3Hで，ファイルをCSAVEで書き込んだファイル(トークン・ファイル)である，と解釈しています．前ページに，そのフィジカルフォーマットを示します．

また,CSAVEの確認(ベリファイ)のための命令としてCLOAD? がありますが,この命令はCLOAD PRINTとしても同じになります．これは，〝?〃と〝PRINT〃の中間言語が同じであるためです．

4-1-2　PRINT #-1(アスキー・ファイル)

データをテープに書き込んだり，読み出したりする命令に，PRINT#-1やINPUT#-1がありま す．このときデータは，アスキー形式に変換され書き込まれます．

アスキー・ファイルはテープの進行方向上に１ビットずつ記録されています．次の図にその様子を示します．

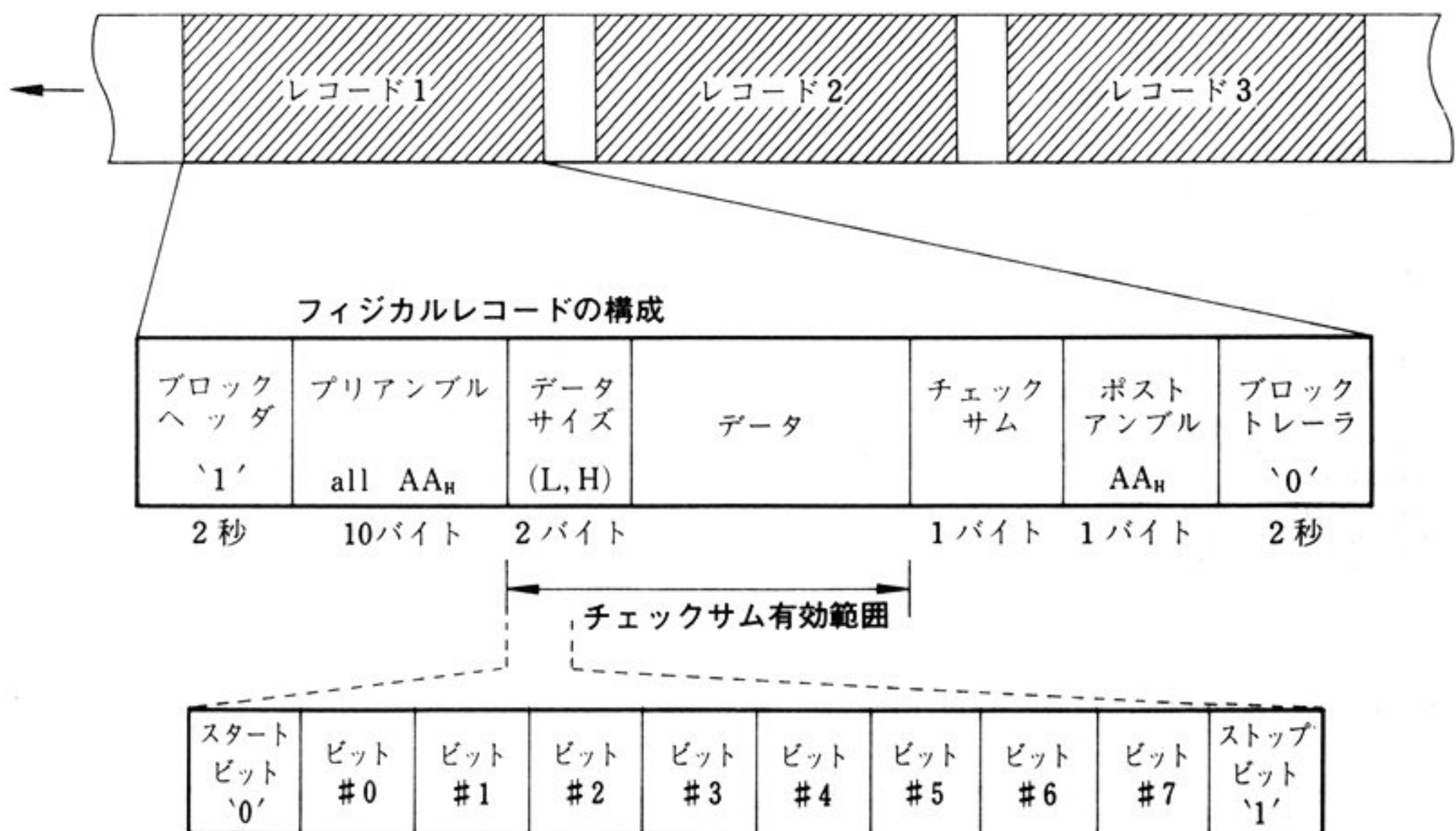



図4―1―3　アスキーファイルレコードフォーマット

ここで，データのテープへの書き込みを試してみましょう．次のプログラムでは素直にデータを書き込んでいます．

```
100 WIDTH 80
110 OVER=TIME
120 FOR I=1 TO 20
130 PRINT I,
140   PRINT#-1,I
150 NEXT I
160 PRINT TIME-OVER;"sec"
```

このプログラムでは，20個の数値をテープに書き込むために約90秒かかりました．テープに書

き込む回数を減らせば時間を短縮できますが,

```
PRINT#-1,1,2,3,4,5,6,7
```

このようにしてカンマで区切っても, カンマ1回ごとにWAIT(テープの音の低い部分)が入るのであまり変わりません.

データを文字列に合成して, 1つの文字変数として書き込めば時間は大幅に短くなります.

次のプログラムでは, データの区切として〝/〞を使っています.

```
10 '**
20 '** CMT WRITE SAMPLE
30 '**
100 WIDTH 80
110 TIME$="00:00:00"
120 OVER=TIME
130 FOR I=1 TO 20
140   A$=A$+STR$(I)+"/"
150   WRITE I,A$
160 NEXT I
170 PRINT#-1,A$:PRINT A$
180 PRINT TIME-OVER
190 '
200 PRINT"Rewind tape and hit any key"
210 A$=INPUT$(1)
220 OVER=TIME
230 INPUT#-1,A$:PRINT A$
240 FOR I=1 TO LEN(A$)
250   C$=MID$(A$,I,1)
260   IF C$="/" THEN PRINT VAL(B$),:B$="" ELSE B$=B$+C$
270 NEXT I
280 PRINT
290 PRINT TIME-OVER
300 END
```

4-1-3 BSAVE, BLOAD(バイナリ・ファイル)

メモリの内容をバイナリ形式でテープに書き込んだり読み出したりするための命令として, BSAVE, BLOADがあります. 形式は次の通りです.

BLOAD#-1, 〝ファイル名〞[, 開始番地]

BSAVE#-1, 〝ファイル名〞, 開始番地, サイズ

BSAVE, BLOADのフォーマットは次のようになります.

0 1 FE_H	2バイト	(BSAVEヘッダーレコード識別子)
ファイル名	6バイト	(6バイトに満たないとき00Hを補う)
スタートアドレス	2バイト	(low, high)
サイズ	2バイト	(low, high)

〈メモリイメージファイルのフィジカルフォーマット〉

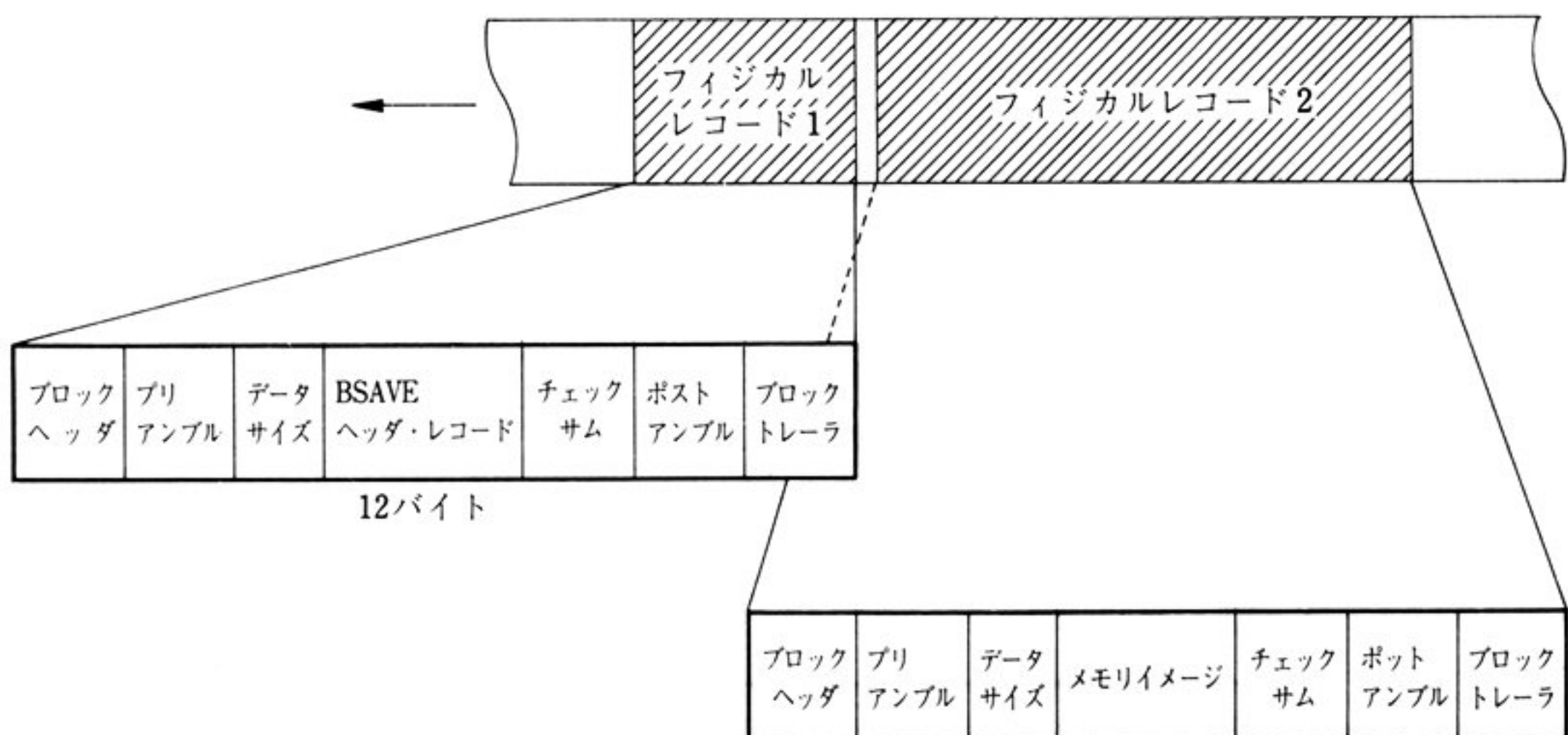



図4－1－4　BSAVEのフォーマット

また，ファイル名にはCLOAD，CSAVE同様，変数を使うことができます．

4-2　フロッピイ・ディスク

4-2-1　ディスク・フォーマット

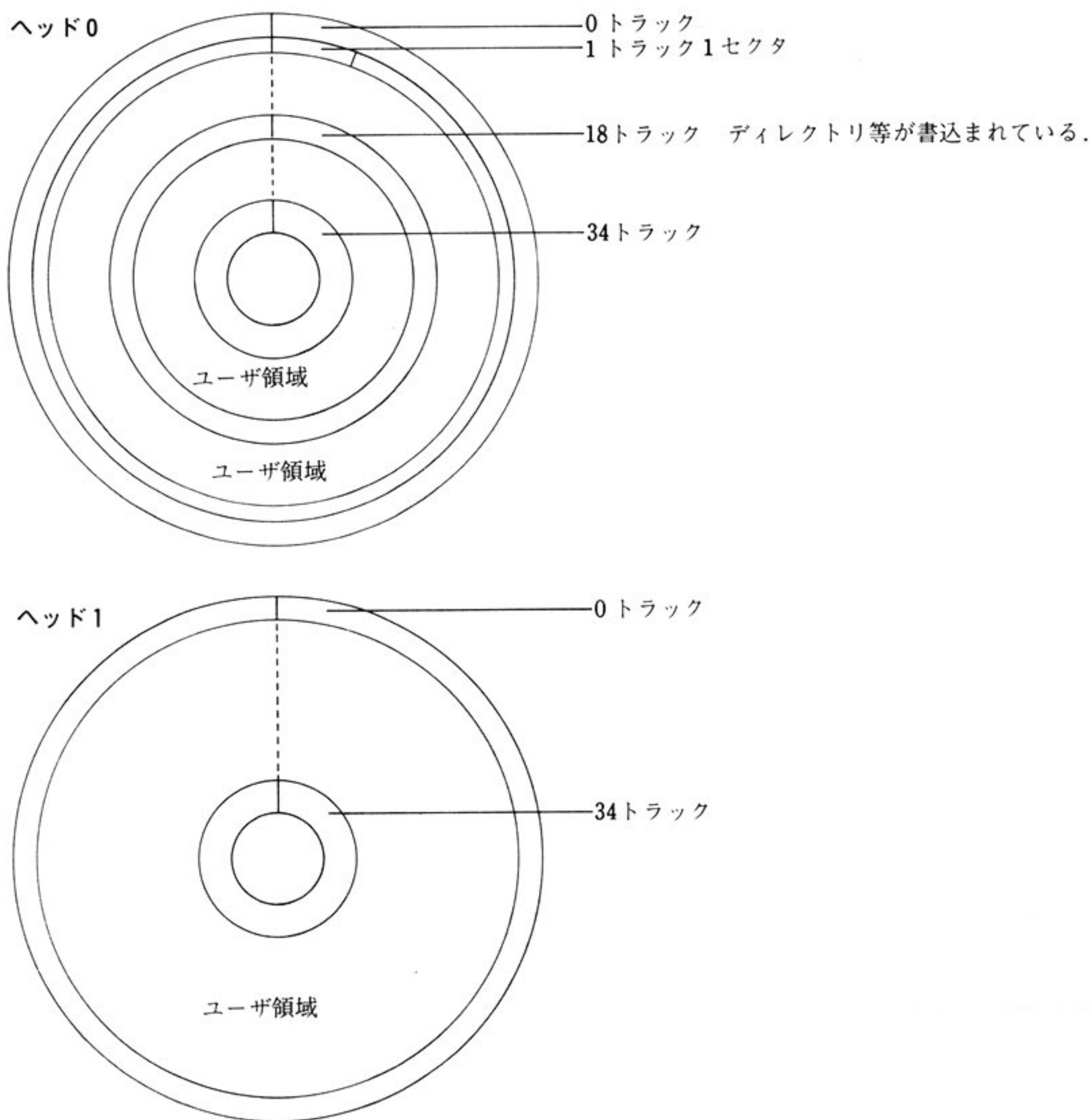
T-DISKBASICでは，ミニフロッピィの入出力の管理はクラスタと呼ばれる，8セクタごとの2Kバイトで行われています．その構成は次のようになっています．

トラック	0～34　(35トラック)
ヘッド	0～1　(2面)
セクタ	1セクタ当り256バイトの倍密度でトラック当り1～16の16セクタ(ただしトラック0　ヘッド0　は単密度(128B))

図4－2－1　ミニフロッピーの構成

アドレッシングは図4-2-2のようになっています.

システムディスクには,漢字パターンを登録したファイルが付属され,BASICから読み出すことができます.

このファイルは,ディレクトリ上では,かくされていて,ファイル名を見ることはできないようになっています.

トラック	ヘッド	セクタ	クラスタ番号	備　　考
0	0	1～8	0	(単密度)
	1	1～8	1	(単密度)
	0	9～16	2	
	1	9～16	3	
1	0	1～8	4	
	1	1～8	5	
	0	9～16	6	
	1	9～16	7	
～	～	～	～	
34	0	1～8	136	
	1	1～8	137	
	0	9～16	138	
	1	9～16	139	

図表4－2－2　アドレッシング

システムディスクと漢字パターンのファイルについては，次のアドレッシングになっています．

図4－2－3　システムと漢字パターンファイルのアドレッシング

トラック	ヘッド	セクタ	
1	0	1～16	T－DISK BASIC システム
	1	1～16	
2	0	1～16	
	1	1～16	
～	～	～	
8	0	1～16	
	1	1～16	

トラック	ヘッド	セクタ	
19	1	1～16	漢字パターンファイル
20	0	1～16	
	1	1～16	
21	0	1～16	
	1	1～16	
～	～	～	
34	0	1～16	
	1	1～16	

かくれた漢字ファイルがあるため，システムディスクの空きエリアは40クラスタしかありません．長いプログラム等のファイルをいくつも書き込むためには，ワークディスクを作るか，漢字ファイルをとる必要があります．(第6章参照)

DISKBASICでは，ディスクの中に書き込まれたファイルの名前を見るときに，ディスク中のディレクトリと呼ばれるテーブルを参照します．

ディレクトリは，18トラック0ヘッドの1～12セクタに書き込まれています．それぞれは16バイトで，次のようになっています．

```
 track 18 head  0 sector  1
00 : 46 44 55 54 49 4C 20 20 20 80 47 FF FF FF FF FF   FDUTIL    _G.....
10 : 46 4F 52 4D 41 54 20 20 20 80 49 FF FF FF FF FF   FORMAT    _I.....
20 : 56 4F 4C 43 4F 50 59 20 20 80 45 FF FF FF FF FF   VOLCOPY   _E.....
30 : 56 4F 4C 43 4F 4D 50 20 20 80 4B FF FF FF FF FF   VOLCOMP   _K.....
40 : 4E 45 4F 4E 20 20 20 20 20 80 41 FF FF FF FF FF   NEON      _A.....
50 : 44 52 41 57 20 20 20 20 20 80 3E FF FF FF FF FF   DRAW      _>.....
60 : 4D 4F 4E 49 54 4F 52 20 20 80 3D FF FF FF FF FF   MONITOR   _=.....
70 : 44 49 53 31 20 20 20 20 20 80 3A FF FF FF FF FF   DIS1      _:.....
80 : 50 52 4F 55 54 20 54 4F 53 80 39 FF FF FF FF FF   PROUT TOS_9.....
90 : 44 49 53 4D 32 2D 30 20 20 80 2A FF FF FF FF FF   DISM2-0   _*.....
A0 : 44 49 53 4D 33 20 20 20 20 80 35 FF FF FF FF FF   DISM3     _5.....
B0 : 4D 4F 4E 31 2D 31 20 20 20 80 2D FF FF FF FF FF   MON1-1    _-.....
C0 : 44 49 53 41 20 20 20 20 20 00 33 FF FF FF FF FF   DISA      .3.....
D0 : 44 49 53 4B 55 54 49 4C 20 00 25 FF FF FF FF FF   DISKUTIL  .%.....
E0 : 45 52 52 4F 52 4D 20 20 20 80 29 FF FF FF FF FF   ERRORM    _).....
F0 : 41 4C 4C 52 41 4D 20 20 20 80 26 FF FF FF FF FF   ALLRAM    _&.....
     └──── ファイル名 ────┘ └拡張子┘ 属性 クラスタ  └─ 未使用 ─┘
                                          番号
```

これらは次のような意味を持っています．

byte 0～8　ファイル名　最大9文字
(00H，FFH，3AH(:)を含んではならない)

9　ファイル属性
00H…ソース形式(JISコード)
10H…ソース形式，書込禁止
01H…機械語プログラム(BSAVE形式)
11H…機械語プログラム・書込禁止
80H…バイナリ形式(中間言語形式)
90H…バイナリ形式・書込禁止

10　クラスタ番号　ファイル領域の先頭クラスタ番号
(FAT内のポインタでもある)

図4－2－4　ディレクトリ

T-DISKBASICでは，次の3種類のミニフロッピィディスクがあります．

・T-DISKBASIC・システムディスク

・システムディスクとして初期化されたディスク

・ユーザ・ディスク用として初期化されたディスク

それぞれのフォーマットは次のようになっています．

図4－2－5　ディスクフォーマット

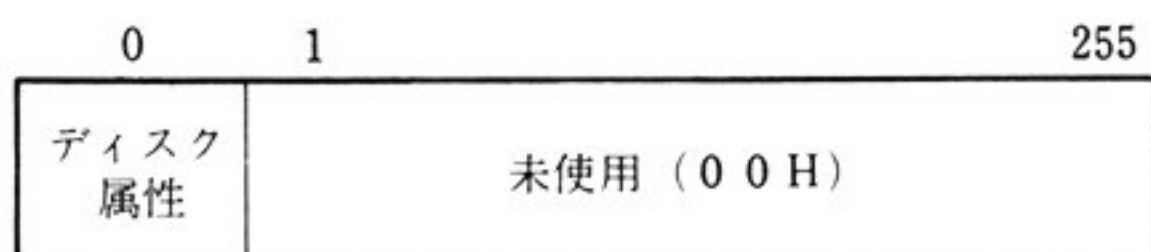
4-2-2　ディスクのダンプ

ディスクの内容を見ようとするとき，単純にA$=DSKI$(1，1，1，1,)とすると，最後の1バイトが欠け，255バイトしかA$に代入されません．256バイト見るためにはFIELD命令で2つの変数に割り当てる必要があります．

次のプログラムを使えばディスクの内容をダンプすることができます.

```
10000 '***
10010 '***          FDDUMP
10020 '***
10030 '
10040 '     If you want to use printer then change printer switch.
10050 '     Printer switch is next line.
10060 PRINTER=0
10070 '     if PRINTER=0 then not use printer
10080 '                    else use printer.
10090  WIDTH 80:KEY OFF:SCREEN 0:CLS
10100 ON ERROR GOTO 10490
10110 PRINT"*** Floppy Disk Dump ***":PRINT
10120 INPUT"Drive  ";DRIVE
10130 INPUT"head   ";HED
10140 INPUT"track  ";TRACK
10150 INPUT"sector ";SECTA :PRINT
10160 FIELD#0,128 AS A$,128 AS B$
10170 FOR SECTOR=SECTA TO 16
10180 DUMMY$=DSKI$(DRIVE,HED,TRACK,SECTOR)
10190 PRINT USING" track ## head ## sector ##";TRACK;HED;SECTOR
10200 IF PRINTER THEN LPRINT USING" track ## head ## sector ##";TRACK;HED;SECTOR
10205 DI$="     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F":PRINT DI$
10210 FOR I=0 TO 7
10220    DI$=HEX$(I)+"0 : "
10230    FOR J=0 TO 15
10240       DI$=DI$+RIGHT$("0"+HEX$(ASC(MID$(A$,I*16+J+1,1))),2)+" "
10250    NEXT J
10260    DI$=DI$+"  "
10270    FOR J=0 TO 15
10280       Z=ASC(MID$(A$,I*16+J+1,1))
10290       IF 31<Z AND Z<248 THEN DI$=DI$+ CHR$(Z) ELSE DI$=DI$+"."
10300    NEXT J
10310    PRINT DI$:IF PRINTER THEN LPRINT DI$
10320 NEXT I
10330 FOR I=8 TO 15
10340    W1=I-8
10350    DI$=HEX$(I)+"0 : "
10360    FOR J=0 TO 15
10370       DI$=DI$+RIGHT$("0"+HEX$(ASC(MID$(B$,W1*16+J+1,1))),2)+" "
10380    NEXT J
10390    DI$=DI$+"  "
10400    FOR J=0 TO 15
10410       Z=ASC(MID$(B$,W1*16+J+1,1))
10420       IF 31<Z AND Z<248 THEN DI$=DI$+CHR$(Z) ELSE DI$=DI$+"."
10430    NEXT J
10440    PRINT DI$:IF PRINTER THEN LPRINT DI$
10450 NEXT I
10460 PRINT:IF PRINTER THEN LPRINT
10470 NEXT SECTOR
10480 END
10490 '
10500 '     Error resume
10510 '
10520 IF ERR=64 THEN PRINT"Bad drive number":RESUME 10120
10530 IF ERR=55 THEN PRINT"Disk I/O error":RESUME 10110
10540 IF ERR=65 THEN PRINT"Bad hed/track/sector":RESUME 10130
10550 ON ERROR GOTO 0
10560 END
```

4-2-3 ディスクの属性

ディスクの属性は18トラック・0ヘッド・13セクタの最初の1バイトによって決まります.

ディスク属性．SET文で設定された属性である．

００Ｈ―属性なし（空白）
１０Ｈ―書き込み禁止（Ｐ）
２０Ｈ―EBCDIC（Ｅ）……未使用
４０Ｈ―リードアフタライト（Ｒ）

図４―２―６　ディスクの属性

属性はSET命令で設定する他に，次のプログラムでも設定できます．次のプログラムを実行すると，書き込みを禁止することができます．

```
10 '**
20 '**  File write protect
30 '**
100 PRINT"set floppy-disk for drive 1 and hit [RETURN] key";
110 LINE INPUT A$
120 FIELD #0,1 AS A$
130 LSET A$="P"
140 DSKO$ 2,0,18,13
```

4-2-4　FAT（ファイル・アロケーション・テーブル）

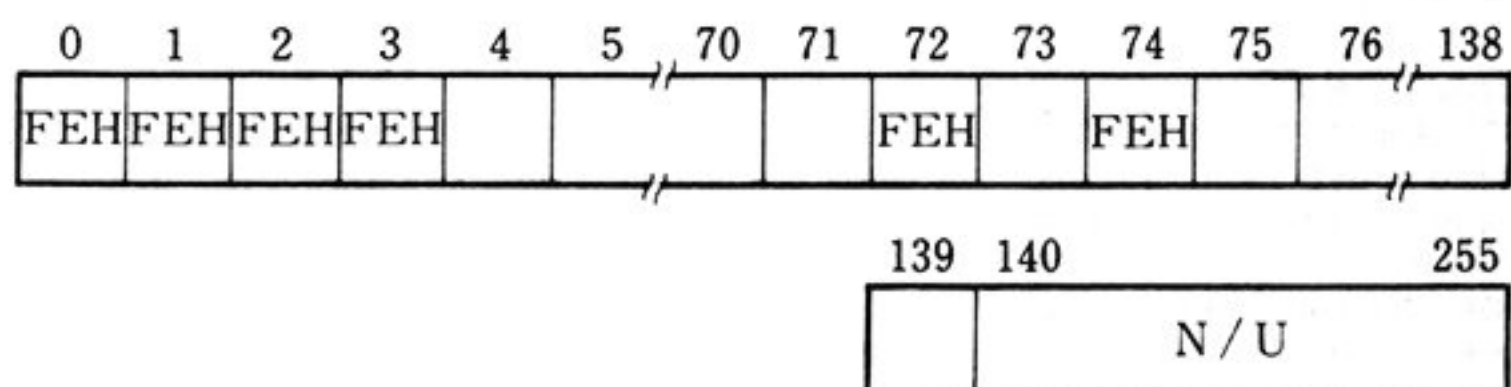
FAT（File　Allocation　Table）はディスク領域の使用状況を表しています．FATは18トラック・０ヘッドの14～16クラスタにあり，これら３セクタの内容は同じもので，15，16は写しです．FATはディスク領域をクラスタ（８セクタ）単位で管理しています．

００Ｈ～８ＢＨ―Nextクラスタ番号（＝FAT内ポインタ）
Ｃ１Ｈ～Ｃ８Ｈ―ラストクラスタ（CXH）
１～８：有効セクタ数
ＦＥＨ―Reserved
・クラスタ０，１，２，３は常に予約済み
・クラスタ72，74は，ディレクトリ，ディスク属性，FATを記録してあるため常に予約済み
ＦＦＨ―未使用

図４―２―７　FAT

T-DISKBASICシステムや漢字パターンを含むディスクではその部分のFATはFEHとなってい

ます．

4-2-5　ディスクへの直接書き込み

プログラムを間違えてKILLしてしまったときや，システムプログラムを一部変更したいとき等，ディスクの内容を直接修正したいときがあります．このときには次のプログラムを使って下さい．このプログラムは，1セクタごとに書き換えをすることができます．

```
1000 '***
1010 '***  FD SCREEN EDITOR FOR T-DISK BASIC
1020 '***
1030 WIDTH 80:SCREEN 0
1040 COLOR 7,0:CLS
1050 KEY OFF
1060 GOSUB 1150 :' CONTROL CODE
1070 GOSUB 1280 :' INPUT
1080 GOSUB 1440 :' DISK READ
1090 GOSUB 1580 :' DUMP
1100 GOSUB 1850 :' EDIT
1110 GOSUB 2300 :' CHANGE SECTOR
1120 IF FLG1 THEN FLG1=0:GOTO 1090
1130 IF FLG2 THEN FLG2=0:GOTO 2640 :' SAVE DISK
1140 IF FLG3 THEN END ELSE 1110
1150 ''''''
1160 '''''' CONTROL CODE
1170 ''''''
1180 CR$=CHR$(13)
1190 L.ARROW$=CHR$(29):R.ARROW$=CHR$(28)
1200 U.ARROW$=CHR$(30):D.ARROW$=CHR$(31)
1210 HOME$=CHR$(11)    :CL$ =CHR$(12)
1220 DEF FNADR(X,Y)=(X-6)/3+(Y-2)*16   :' get adreess
1230 YMAX=17:YMIN=2
1240 XMAX=52:XMIN=6
1250 C4$=CHR$(252):C5$=CHR$(253):C6$=CHR$(254):C7$=CHR$(255)
1260 FLG1=0:FLG2=0:FLG3=0
1270 RETURN
1280 '''''
1290 ''''' INPUT DUMP LOCATION
1300 '''''
1310 PRINT TAB(25);"<<<` FD EDITOR   >>>
1320 PRINT
1330 INPUT" INPUT DRIVE    :",DR
1340 IF DR<1 OR DR>4 THEN 1330
1350 INPUT" INPUT HED      :",HE
1360 IF H<>0 AND H<>1 THEN 1350
1370 INPUT" INPUT TRACK    :",TR
1380 IF TR<0 OR TR>34 THEN 1370
1390 INPUT" INPUT START SECTOR :";STSEC
1400 IF STSEC<1 OR STSEC>16 THEN 1390
1410 INPUT" INPUT END SECTOR( < [START SECTOR]+8):";ENDSEC
1420 IF ENDSEC<STSEC OR ENDSEC>STSEC+7 OR ENDSEC>16 THEN 1410
1430 RETURN
1440 '''''
1450 '''''  READ DISK
1460 '''''
1470 DIM BUFFER$(1,7)
1480 NBUFFER=STSEC-ENDSEC+1
1490 FIELD #0,128 AS F1$,128 AS F2$
1500 FOR I=STSEC TO ENDSEC
1510   DUMMY$=DSKI$(DR,HE,TR,I)
1520   BUFFER$(0,I-STSEC)=F1$
1530   BUFFER$(1,I-STSEC)=F2$
1540 NEXT I
1550 BUFFER.NUM=0:BUFFER.NUMP=ENDSEC-STSEC
1560 GOSUB 2550 :' MOVE BUFFER
1570 RETURN
1580 '''''
```

```
1590 '''''' DUMP
1600 ''''''
1610 GOSUB 2550
1620 BUL=PEEK(VARPTR(F1$)+1)
1630 BUH=PEEK(VARPTR(F1$)+2)
1640 BUFFER=BUL+BUH*256
1650 CLS
1660 LOCATE 0,0:PRINT "<<< FD EDITER >>>";
1670 PRINT TAB(20);C5$;:PRINT USING"DRIVE=# TRACK=## HED=# SECTOR=##";DR;TR;HE;S
TSEC+BUFFER.NUM
1680 GOSUB 2130
1690 LOCATE 0,1:PRINT"----+--0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-+ 01
23456789ABCDEF "
1700 FOR I=2 TO 17
1710   LOCATE 4,I:PRINT"I";TAB(54);"I";
1720 NEXT I
1730 PRINT"----+--0--1--2--3--4--5--6--7--8--9--A--B--C--D--E--F-+ 0123456789ABC
DEF "
1740 LOCATE 68,0:PRINT C6$;"BUFFER:";BUFFER.NUM
1750 FOR I=0 TO 15
1760   LOCATE 0,I+2:PRINT RIGHT$("0"+HEX$(I*16),2);
1770   FOR J=0 TO 15
1780     M=PEEK(BUFFER+I*16+J)
1790     LOCATE J*3+6,I+2:PRINT RIGHT$("0"+HEX$(M),2);
1800     LOCATE J+56,I+2
1810     IF M<32 OR M>128 THEN PRINT"."; ELSE PRINT CHR$(M);
1820   NEXT J
1830 NEXT I
1840 RETURN
1850 ''''''
1860 '''''' EDIT
1870 ''''''
1880 X=XMIN:Y=YMIN
1890 '
1900 '
1910   LOCATE X,Y
1920   A$=INPUT$(1):SOUND 50,2
1930   IF A$=CR$ THEN RETURN
1940   IF A$=D.ARROW$ THEN Y=Y+1:IF Y>YMAX THEN Y=YMIN
1950   IF A$=U.ARROW$ THEN Y=Y-1:IF Y<YMIN THEN Y=YMAX
1960   IF A$=R.ARROW$ THEN X=X+1:IF X>XMAX THEN X=XMIN:Y=Y+1:IF Y>YMAX THEN Y=Y-
1 ELSE 1970 ELSE IF SCREEN(X,Y)=32 THEN 1960
1970   IF A$=L.ARROW$ THEN X=X-1:IF X<XMIN THEN X=XMAX:Y=Y-1:IF Y<YMIN THEN Y=Y+
1 ELSE 1980 ELSE IF SCREEN(X,Y)=32 THEN 1970
1980   IF A$=HOME$ OR A$=CL$ THEN X=XMIN:Y=YMIN
1990   IF (A$>="0" AND A$<="9") OR (A$>="A" AND A$<="F") OR (A$>="a" AND A$<="f"
) THEN IF SCREEN(X,Y)<>32 THEN GOSUB 2010:A$=R.ARROW$:GOTO 1960
2000 GOTO 1910
2010 ''''''
2020 '''''' CHANGE VAL
2030 ''''''
2040 LOCATE X,Y :PRINT A$
2050 X1=(X¥3)*3
2060 A$=CHR$(SCREEN(X1,Y))+CHR$(SCREEN(X1+1,Y))
2070 M=VAL("&H"+A$)
2080 ADRS=FNADR(X1,Y)+BUFFER
2090 POKE ADRS,M
2100 LOCATE 56+FNADR(X,Y) MOD 16,Y
2110 IF M>31 AND M<128 THEN PRINT CHR$(M); ELSE PRINT ".";
2120 RETURN
2130 ''''''
2140 '''''' INFORMATION
2150 ''''''
2160 GOSUB 2460
2170 COLOR 7,0:LOCATE 15,19
2180 PRINT C4$;"---- FD EDITOR has following commands ----"
2190 LOCATE 0,20
2200 PRINT C4$;"    key       : function"
2210 PRINT C5$;"upper arrow :";C6$;"coursol up.    "
2220 PRINT C5$;"down  arrow :";C6$;"coursol down.  "
2230 PRINT C5$;"right arrow :";C6$;"coursol right. "
2240 PRINT C5$;"left  arrow :";C6$;"coursol left.  ";
2250 LOCATE 30,20:PRINT C4$;" Key       : function"
```

```
2260 LOCATE 30,21:PRINT C5$;" cls/home :";C6$;"coursol move to home position."
2270 LOCATE 30,22:PRINT C5$;" return   :";C6$;"end of edit.                 "
2280 LOCATE 30,23:PRINT C5$;" ctrl-s   :";C6$;"key buffer clear.            "
2290 RETURN
2300 ''''''
2310 '''''' CHANGE SECTOR
2320 ''''''
2330 GOSUB 2460
2340 PRINT TAB(25);C4$;"-----  MENU -----"
2350 PRINT TAB(5)  ;" 1 : EDIT NEXT SECTOR";
2360 PRINT TAB(45);" 2 : EDIT OTHER SECTOR"
2370 PRINT TAB(5)  ;" 3 : SAVE FOR DISK ";
2380 PRINT TAB(45);" 4 : END OF PROGRAM   "
2390 INPUT "Which do you select :";W$
2400 IF LEN(W$)<>1 OR W$<"1" OR W$>"4" THEN 2330
2410 IF W$="1" THEN IF BUFFER.NUM<6 THEN BUFFER.NUM=BUFFER.NUM+1 :FLG1=1:ELSE LO
CATE 0,22:PRINT C6$;"ERROR -- NEXT SECTOR IS NOT ON MEMORY ":GOTO 2330
2420 IF W$="2" THEN LOCATE 0,22:PRINT C7$;"INPUT NUMBER OF BUFFER ";:INPUT  A$:B
UFFER.NUM=VAL(A$):FLG1=1
2430 IF W$="3" THEN PRINT C5$;"SURE (Y/N)";:INPUT A$:IF A$="Y" OR A$="y" OR A$="
"THEN FLG2=1:RETURN ELSE 2330
2440 IF W$="4" THEN PRINT"END OK(Y/N)";:INPUT A$:IF A$="Y" OR A$="y" OR A$="" TH
EN FLG3=1:RETURN ELSE 2330
2450 RETURN
2460 ''''''
2470 '''''' CLEAR FOR LINE 19-24
2480 ''''''
2490 FOR I=19 TO 24
2500 LOCATE 0,I
2510   PRINT TAB(78);
2520 NEXT I
2530 LOCATE 0,19
2540 RETURN
2550 ''''''
2560 ''''''  MOVE BUFFER
2570 ''''''
2580 BUFFER$(0,BUFFER.NUMP)=F1$
2590 BUFFER$(1,BUFFER.NUMP)=F2$
2600 BUFFER.NUMP=BUFFER.NUM
2610 LSET F1$=BUFFER$(0,BUFFER.NUM)
2620 LSET F2$=BUFFER$(1,BUFFER.NUM)
2630 RETURN
2640 ''''''
2650 '''''' SAVE TO DISK
2660 ''''''
2670 GOSUB 2550
2680 FOR I=STSEC TO ENDSEC
2690   LSET F1$=BUFFER$(0,I-STSEC)
2700   LSET F2$=BUFFER$(1,I-STSEC)
2710   DSKO$ DR,HE,TR,I
2720 NEXT I
2730 GOSUB 2460
2740 LOCATE 0,20:PRINT "SAVE OK"
2750 INPUT "DO YOU WANT TO EDIT ANOTHER SECTOR?";A$
2760 IF A$="Y" OR A$="y".THEN RUN
2770 END
```

4-2-6　データ・ディスクをシステム・ディスクに

データ・ディスクをファイルを壊さずシステム・ディスクにするためには，4～35クラスタにあるシステムをディスクに書き込み，FATを書き換えておけばよいのです．

実際の手順を次に示します．

1) ドライブ1にシステムディスクを，ドライブ2にデータディスクを入れる．

2) ドライブ1がシステムディスクかどうか確認する．

3)データディスクのFATを調べる，システムを書き込めるかどうかを確認する．

4)システムディスクの４クラスタから35クラスタを，データディスクに書き込む．

5)システムの書き込まれた部分のFATを書き換えて予約済(FEH)とする．

次のプログラムではこの手順に従ってシステムをコピーしています．

```
10 '***
20 '***   SYSTEM DISK GENERATOR
30 '***
100 COLOR 7,0:WIDTH 80:SCREEN 0:CLS
110 PRINT TAB(20);"<<< SYSTEM DISK GENERATOR ver 1.0 >>>"
120 PRINT
130 PRINT TAB(25);"MOUNT SYSTEM DISK FOR DRIVE 1"
140 PRINT TAB(25);"MOUNT  DATA  DISK FOR DRIVE 2"
150 PRINT TAB(25);"                OK? (Y/N)     "
160 A$=INKEY$:IF A$="Y" OR A$="y" OR A$=CHR$(13) THEN 170 ELSE IF A$="N" OR A$="
n" THEN END ELSE 160
170 '
180 BUFFERST=&H96CF
190 '
200 '
210 ' CHECK SYSTEM FD1
220 '
230 FIELD#0,1 AS A$
240 DUMMY$=DSKI$(1,0,1,1)
250 IF A$<>CHR$(&HF3) THEN PRINT"DR.1 IS NOT SYSTEM DISK! ":END
290 '
300 ' CHECK FAT ( CAN GENERATE SYSTEM ? )
310 '
320 FIELD#0,139 AS A$
330 DUMMY$=DSKI$(2,0,18,14)
340 FOR I=5 TO 32
350   IF ASC(MID$(A$,I,1))<>&HFF THEN PRINT"CAN'T MAKE SYSTEM ":SOUND 60,20:END
370 NEXT
380 '
390 ' SYSTEM COPY ( FROM FD1 TO FD2 )
400 '
410 FIELD #0,128 AS A$,128 AS B$
420 FOR TR=1 TO 8
430   FOR HED=0 TO 1
440     FOR SEC=1 TO 16
450       LOCATE 10,10:PRINT USING"NOW COPY # HED # TRACK ## SECTOR";HED;TR;SEC
460       DUMMY$=DSKI$(1,HED,TR,SEC)
470       DSKO$ 2,HED,TR,SEC
480       CH1$=A$:CH2$=B$:LOCATE 20,12:PRINT TAB(40);
490       DUMMY$=DSKI$(2,HED,TR,SEC)
500       LOCATE 20,12
510       IF CH1$=A$ AND CH2$=B$ THEN PRINT "CHECH OK!           " ELSE PRINT"CHE
CK NG! RETRING":GOTO 450
520     NEXT
530   NEXT
540 NEXT
560 '
570 ' WRITE FAT FD2
580 '
600 DUMMY$=DSKI$(2,0,18,14)
610 FOR I=5 TO 32
620   POKE BUFFERST+I-1,&HFE
630 NEXT
640 FOR I=14 TO 16
650 DSKO$ 2,0,18,I
660 NEXT
700 END
```

4-3 RAMPAC

4-3-1 データ・フォーマット

RAMPAC 2の入出力処理はセクタ単位になっています．4KのRAMPACでは，次のようになっています．

トラック　：0～3の4トラック

セクタ　：1～4の4セクタ

アドレッシングは次のようになります．

トラック	0	セクタ	1～2	：ディレクトリ
		セクタ	3	：パック2の属性
		セクタ	4	：FAT
トラック	1	セクタ	1～4	：ユーザ・エリア
	2	セクタ	1～4	：ユーザ・エリア
	3	セクタ	1～4	：ユーザ・エリア(計3Kバイト)

4-3-2 ダンプ・プログラム

RAMPAC 2のダンププログラムを次に示します．

```
100 '***
110 '***  Damp program for RAM PACK 2
120 '***               ( T-basic )
130 WIDTH 80:KEY OFF:SCREEN 1:CLS
140 PRINT"*** RAM.PAC 2 Dump ***":PRINT
150 INPUT"track (0-3)";TRACK
160 INPUT"sector(1-4)";BEGIN
170 FIELD#0,128 AS A$,128 AS B$
180 FOR SECTOR=BEGIN TO 4
190   PRINT"TRACK=";TRACK,"SECTOR=";SECTOR
200   W1$=DSKI$(5,TRACK,SECTOR)
210   FOR I=0 TO 7
220     PRINT HEX$(I);"0 : ";
230     FOR J=0 TO 15
240       PRINT USING"&& ";RIGHT$("0"+HEX$(ASC(MID$(A$,I*16+J+1,1))),2);
250     NEXT J
260     PRINT"  ";
270     FOR J=0 TO 15
280       Z=ASC(MID$(A$,I*16+J+1,1))
290       IF 31<Z AND Z<248 THEN PRINT CHR$(Z); ELSE PRINT".";
300     NEXT J
310     PRINT
320   NEXT I
330   FOR I=8 TO 15
340     W1=I-8
350     PRINT HEX$(I);"0 : ";
360       FOR J=0 TO 15
370         PRINT USING"&& ";RIGHT$("0"+HEX$(ASC(MID$(B$,W1*16+J+1,1))),2);
380       NEXT J
390     PRINT"  ";
400       FOR J=0 TO 15
410         Z=ASC(MID$(B$,W1*16+J+1,1))
420         IF 31<Z AND Z<248 THEN PRINT CHR$(Z); ELSE PRINT".";
430       NEXT J
```

```
440       PRINT
450     NEXT I
460     PRINT
470 NEXT SECTOR
480 GOTO 150
```

このプログラムは，DISKBASICなしで走らせることができます．RAMPAC関係の命令はROM版のT-BASICでサポートされているからです．

RUNしてダンプしたいトラックとダンプ開始セクタを入力すればダンプを開始します．入力エラーのチェックは行っていないので必要なら追加して下さい．

なお，32キロバイトのRAMPACではトラック番号が０～31になります．

4-3-3　ディレクトリ

RAMPACのディレクトリはトラック０，セクタ１～２に格納され，その内容はファイル名９バイト・属性１バイト・クラスタ番号１バイト・未使用５バイトの16バイトとなっています．

RAMパックのクラスタ番号はフロッピーディスクと異り，セクタをトラックの順に数えていったもので，次のように表すことができます．

（クラスタ）＝（トラック）×４＋（セクタ）

ディレクトリは，実際には次のように書き込まれています．属性の意味などについてはディスクの項を見て下さい．

```
TRACK= 0        SECTOR= 1
00 : 55 54 49 4C 32 20 20 20 20 00 04 FF FF FF FF FF    UTIL2     .......
10 : 55 54 49 4C 31 20 20 20 20 80 0B FF FF FF FF FF    UTIL1     _......
20 : 4D 4F 4E 20 20 20 20 20 20 80 12 FF FF FF FF FF    MON       _......
30 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
40 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
50 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
60 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
70 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
80 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
90 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
A0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
B0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
C0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
D0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
E0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
F0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
```

０トラック，３セクタにはRAMPACの属性が格納されています．１バイトの数値によって属性が決まります．この意味はディスクの場合と同様です．

０トラック，４セクタにはFATが格納されています．これは，次のようになっています．

```
TRACK= 0        SECTOR= 4
00 : FE FE FE FE 05 06 07 08 C1 FF FF 0C 0D 0E 0F 10    ........ｼ.......
10 : 11 C1 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20    .ｼ..............
20 : C1 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ｼ...............
30 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
```

4-4　プリンタ出力

4-4-1　LPRINT・PRINT＃-2

プリンタに出力するための命令としては，LPRINTとPRINT#-2があります．

出力データはパソピア内部でバッファリングされることはなく，直接プリンタに出力され，次のときにCR/LFの制御コードが出力されます．

1)LPRINT文の終わりに；がない

2)WIDTH LPRINTで指定した一定量のデータ出力後

2)の場合，出力データ量の計算には文字コード以外の制御コードも計算していますので注意して下さい．なお，WIDTH　LPRINT　255を実行すると出力データ量の計算が無効になります．

4-4-2　ハードコピー

画面上のテキストやグラフィックはCOPYキーを押せばドットプリンタIIにハードコピーをとることができます．

このハードコピーのフックがFFF7Hにありますので，ここをBASICや，機械語プログラムからコールすれば，画面表示をプリンタにハードコピーすることができます．(ただし，レジスタは保存されません)

COPYキーはこのフックに使用されているので，このFFF7Hのフックを書き換えれば，COPYキーを他の目的(機械語サブルーチンの呼出等)に使うこともできます．

次のプログラムはこのフックの使用例です．

```
10 '**
20 '**   HARD COPY ROUTINE CALL
30 '**
40 '*
50 '* CALL HCOPY
60 '*       HCOPY ADDRESS ... &HFFF7
100 ' ADDRESS SET
110 '
120 HCOPY=&HFFF7
130 WIDTH 80
140 CLS
150 FOR I=5 TO 23
160   LOCATE 5,I
170   PRINT STRING$(70,"#")
180 NEXT
190 LOCATE 10,2
200 INPUT "*** HARD COPY Y/N ***";K$
210 IF K$="N" THEN 230
220 CALL HCOPY
230 END
```

```
    *** HARD COPY Y/N ***? y

################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
################################################################################
```

4-4-3 LPRINT・サブルーチン

ドットプリンタII等のプリンタは数多くの機能を持っていて，コントロールコードを覚えて使いこなすことはなかなか大変です．

そこで，コマンドを与えれば，ドットプリンタIIに対してコントロールコードを出力するサブルーチンを作ってみました．

```
10000 '**
10010 '**          LPRINT SUBROUTINE FOR DOT-PRINTER ][
10020 '**
10030 '   INPUT PR.FUNC$  AS PRINTER FUNCTION
10040 '   destroy val : esc$ for CHR$(&h1H)
10050 '   STATUS is error flug .
10060 '             STATUS=0    --- no error
10070 '             STATUS=1    --- syntax error
10080 '             STATUS=2    --- illegal function
10090 IF STATUS THEN SOUND 40,10:PRINT"STATUS=";ST
10100 STATUS=0    :' initilise status
10110 GOSUB 10140
10120 IF STATUS=0 THEN PR.FUNC$=""
10130 RETURN
10140 IF PR.FUNC$="CR"  THEN LPRINT CHR$(&HD); :RETURN:' carrige return
10150 IF PR.FUNC$="LF"  THEN LPRINT CHR$(&HA); :RETURN:' line feed
10160 IF PR.FUNC$="VT"  THEN LPRINT CHR$(&HB); :RETURN:' vertical tab
10170 IF PR.FUNC$="FF"  THEN LPRINT CHR$(&HC); :RETURN:' form feed
10180 IF PR.FUNC$="CAN" THEN LPRINT CHR$(&H18);:RETURN:' cancel
10190 IF PR.FUNC$="SO"  THEN LPRINT CHR$(&HE); :RETURN:' shift out
10200 IF PR.FUNC$="SI"  THEN LPRINT CHR$(&HF); :RETURN:' shift in
10210 IF PR.FUNC$="DC1" THEN LPRINT CHR$(&H11);:RETURN:' device control 1
10220 IF PR.FUNC$="DC3" THEN LPRINT CHR$(&H13);:RETURN:'                3
10230 IF PR.FUNC$="BS"  THEN LPRINT CHR$(&H8); :RETURN:' back space
10240 IF PR.FUNC$="HT"  THEN LPRINT CHR$(&H9); :RETURN:' horisontal tab
10250 IF PR.FUNC$="GS"  THEN LPRINT CHR$(&H1E);:RETURN:' group separator
10260 IF PR.FUNC$="US"  THEN LPRINT CHR$(&H1F);:RETURN:' unit separator
10270 ESC$=CHR$(&H1B)
10280 IF PR.FUNC$="PAICA" THEN LPRINT ESC$;"N";:RETURN:' paica print (10 cpi)
10290 IF PR.FUNC$="PROP"  THEN LPRINT ESC$;"P";:RETURN:' proportional print
10300 IF PR.FUNC$="MIN"   THEN LPRINT ESC$;"Q";:RETURN:' 17 cpi print
10310 IF PR.FUNC$="ELITE" THEN LPRINT ESC$;"E";:RETURN:' elite print (12 cpi)
10320 IF PR.FUNC$="1DOT"  THEN LPRINT ESC$;"1";:RETURN:' 1 dot space
10330 IF PR.FUNC$="2DOT"  THEN LPRINT ESC$,"2";:RETURN:' 2
10340 IF PR.FUNC$="3DOT"  THEN LPRINT ESC$;"3";:RETURN:'
10350 IF PR.FUNC$="4DOT"  THEN LPRINT ESC$;"4";:RETURN
```

```
10360 IF PR.FUNC$="5DOT"  THEN LRRINT ESC$;"5";:RETURN
10370 IF PR.FUNC$="6DOT"  THEN LPRINT ESC$;"6";:RETURN
10380 IF PR.FUNC$="HIRA"  THEN LPRINT ESC$;"&";:RETURN:' hiragana
10390 IF PR.FUNC$="KATA"  THEN LPRINT ESC$;"$";:RETURN:' Katakana
10400 IF PR.FUNC$="NGRA"  THEN LPRINT ESC$;"#";:RETURN:' naibu graphic
10410 IF PR.FUNC$="DGRA"  THEN LPRINT ESC$;"S";N:RETURN:' dot graphic
10420 IF PR.FUNC$="!ON"   THEN LPRINT ESC$;"!";:RETURN:' Kyoutyou moji sitei
10430 IF PR.FUNC$="!OFF"  THEN LPRINT ESC$;CHR$(&H22);:RETURN:'          off
10440 IF PR.FUNC$="INC"   THEN LPRINT ESC$;"[";:RETURN:' incremental mode
10450 IF PR.FUNC$="LOG"   THEN LPRINT ESC$;"]";:RETURN:' logical seek mode
10460 IF PR.FUNC$="1/6"   THEN LPRINT ESC$;"A";:RETURN:' 1/6" Kaigyo
10470 IF PR.FUNC$="1/8"   THEN LPRINT ESC$;"B";:RETURN:' 1/8" Kaigyo
10480 IF PR.FUNC$="N/144" THEN IF N<1 OR N>99 THEN STATUS=2:RETURN ELSE LPRINT E
SC$;"T";N :RETURN                                        :' N/144" kaigyo
10490 IF PR.FUNC$="FOWARD"   THEN LPRINT ESC$;"f";:RETURN:'foward kaigyo
10500 IF LEFT$(PR.FUNC$,3)="REV" THEN LPRINT ESC$;"r";:RETURN:' reverse kaigyo
10510 IF PR.FUNC$="HTSET"   THEN LPRINT ESC$;"(";:RETURN:' horizontal tab set
10520 IF PR.FUNC$="HTRESET" THEN LPRINT ESC$;")";:RETURN:'                reset
10530 IF PR.FUNC$="HTCLR"   THEN LPRINT ESC$;"0";:RETURN:'                all clear
10540 IF PR.FUNC$="UON"     THEN LPRINT ESC$;"X";:RETURN:' under line on
10550 IF PR.FUNC$="UOFF"    THEN LPRINT ESC$;"Y";:RETURN:'                off
10560 IF PR.FUNC$="LEFT" THEN IF N<0 THEN STATUS=2:RETURN ELSE LPRINT ESC$;"L";N
:RETURN                                                  :' left margin set
10570 IF PR.FNNC$="1WAY"    THEN LPRINT ESC$;">";:RETURN:' katahoukow inji
10580 IF PR.FUNC$="2WAY"    THEN LPRINT ESC$;"<";:RETURN:' ryouhoukou inji
10590 STATUS=1:RETURN
```

PR.FUNC$に命令を入れてGOSUB10000とすれば，PR.FUNC$の内容に対応するコントロールコードが出力されます．

例えば，PR.FUNC$＝"FF"ならば，フォームフィードを行います．

印字をパイカにしたければ，PR.FUNC$＝"PAICA"，プロポーショナルにしたければ，PR.PUNC$＝"PROP"としてサブルーチンを呼べばよいのです．

"N/144"でレフトマージンをセットし，ドットグラフィック印字では，Nにそれぞれ改行幅，印字開始位置，バイト数を入れてからサブルーチンを呼んで下さい．

4-4-4　プリンタIIの逆スクロール機能を使う

プリンタIIの逆スクロールを利用した同じページ内の任意の場所に出力するサブルーチンを紹介します．

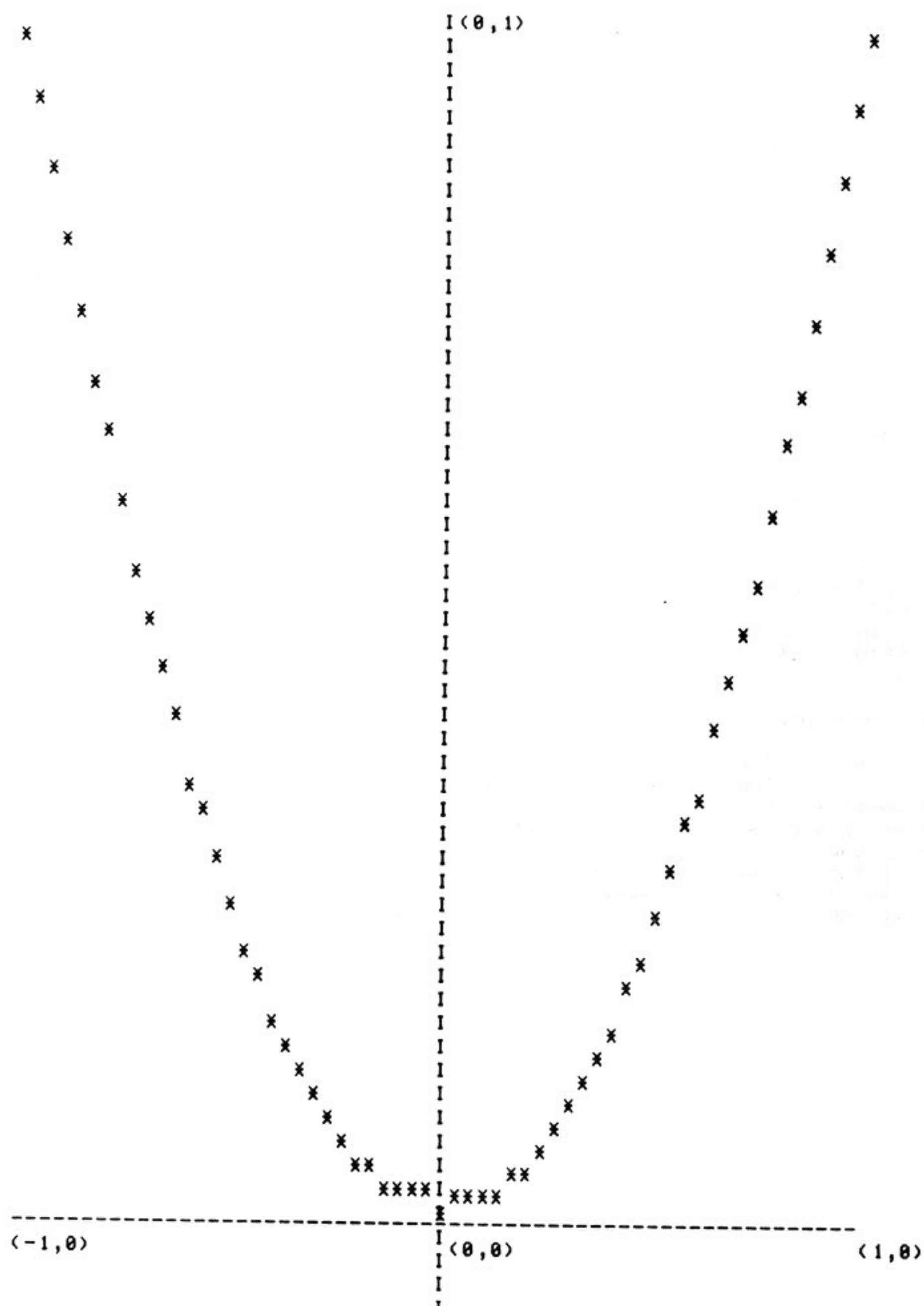
80行から130行の設定ルーチンと1000行からのサブルーチンがそうです．X，Yにそれぞれ行と列を入れ，CHAR$に書きたいキャラクタを入れてサブルーチンコールして下さい．

また，このサブルーチンを利用してグラフを書かせてみました．

必ずプリンタのスイッチをONにして改ページしてから，実行して下さい．そうしないとホームポジションがずれてしまい，出力が狂います．また，逆スクロールさせるときには負担がかかりますので，あまり出力済の紙をためないようにして下さい．

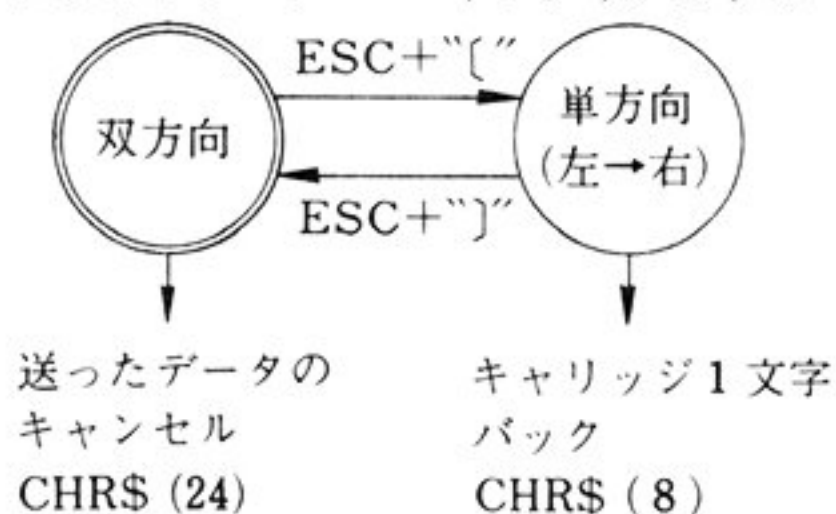
```
10 '***
20 '***  DOT PRINTER II SAMPLE
30 '***
40 '
50 WIDTH LPRINT 255
60 DEF FNF(X)=X*X                 :' DEFINE PLOT FUNCTION
70 '
```

```
80 ' DEFINE PRINTER FUNCTION
90 '
100 LFOWARD$=CHR$(&H1B)+"f"
110 LREVERSE$=CHR$(&H1B)+"r"
120 CR$=CHR$(13):LF$=CHR$(10)
130 LMAX=70
140 '
150 ' INITIAL OUTPUT CRT
160 '
170 WIDTH 80:COLOR 7,0:CLS
180 PRINT TAB(20);"<< DOT PRINTER II SAMPLE PROGRAM >>>"
190 PRINT
200 INPUT"SET PRINTER AND HIT [RETURN] KEY";W$
210 PRINT:PRINT
220 '
230 ' START
240 '
250 X=20:Y=2:CHAR$="<< X-Y PLOTTER SAMPLE >>"
260 GOSUB 1000
270 '
280 ' WAKU
290 '
300 Y=10:X=41:CHAR$="(0,1)":GOSUB 1000
310 X=40:FOR Y=10 TO 65:CHAR$="I":GOSUB 1000:NEXT
320 X=41:Y=61:CHAR$="(0,0)":GOSUB 1000
330 X=10:Y=61:CHAR$="(-1,0)":GOSUB 1000
340 X=10:Y=60:CHAR$=STRING$(60,"-"):GOSUB 1000
350 X=70:Y=61:CHAR$="(1,0)":GOSUB 1000
360 '
370 '
380 ' MAKE FIG.
390 '
400 FOR X=10 TO 70
410   X1=(X-40)/30
420   Y1=FNF(X1)
430   Y=INT(60-50*Y1)
440   CHAR$="*"
450   GOSUB 1006
460 NEXT
470 LPRINT LFOWARD$
480 LPRINT CHR$(12)
490 END
1000 '
1010 ' MOVE HED FOR X,Y & PRINTOUT CHAR$
1020 '
1030 IF HEDY=Y THEN 1090
1040 IF Y>HEDY THEN YDIR=1 ELSE YDIR=-1
1050 W=HEDY+YDIR:IF W<1 OR W>LMAX THEN 1100
1060 IF YDIR>0 THEN LPRINT LFOWARD$;LF$; ELSE LPRINT LREVERSE$;LF$;
1070 HEDY=W
1080 IF INT(Y)<>HEDY THEN 1050
1090 LPRINT CR$;TAB(X);CHAR$;
1100 RETURN
```

〈出力例〉

4-4-5　プリンタ機能一覧表

プリンタ機能一覧表（ドットプリンタⅡ）

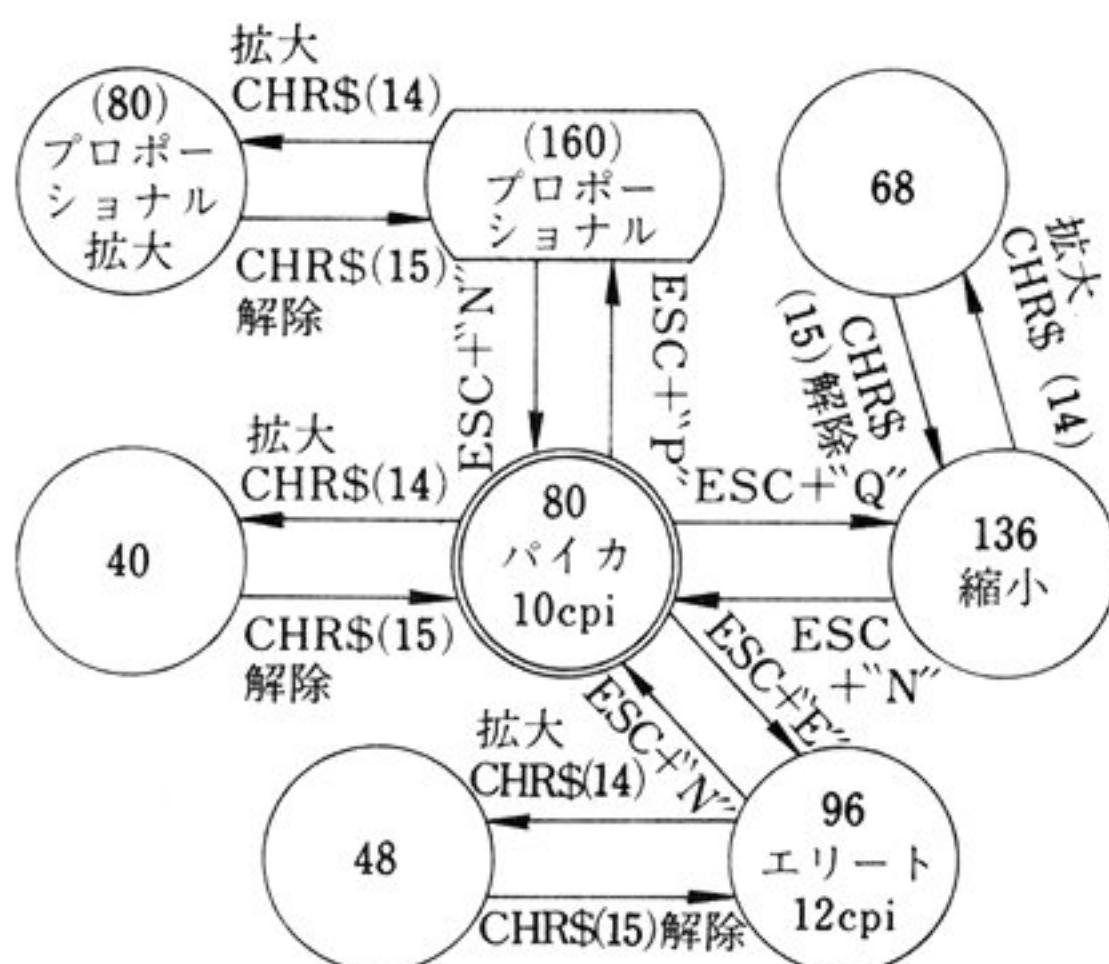
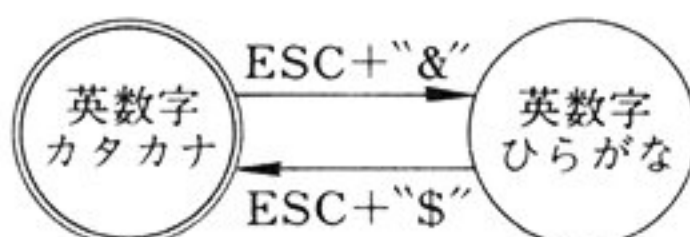
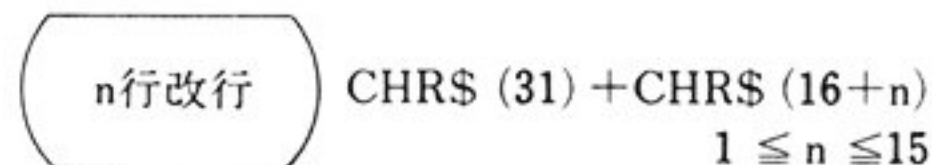
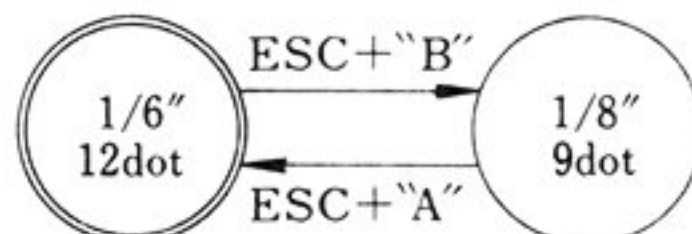
ESC：CHR$(27)
CR：CHR$(13)印字
LF：CHR$(10)印字・改行
FF：CHR$(12)ページ送り

印字方向

印字文字の切換え

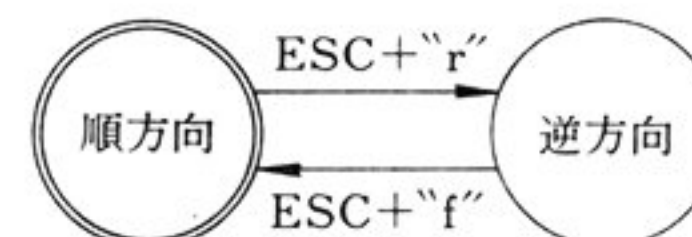
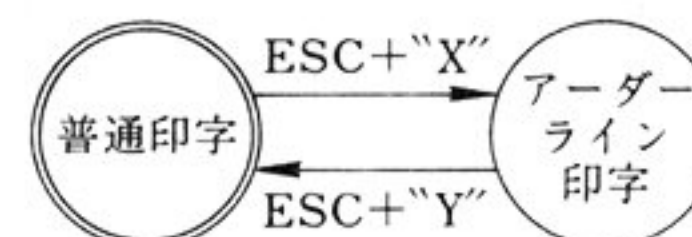
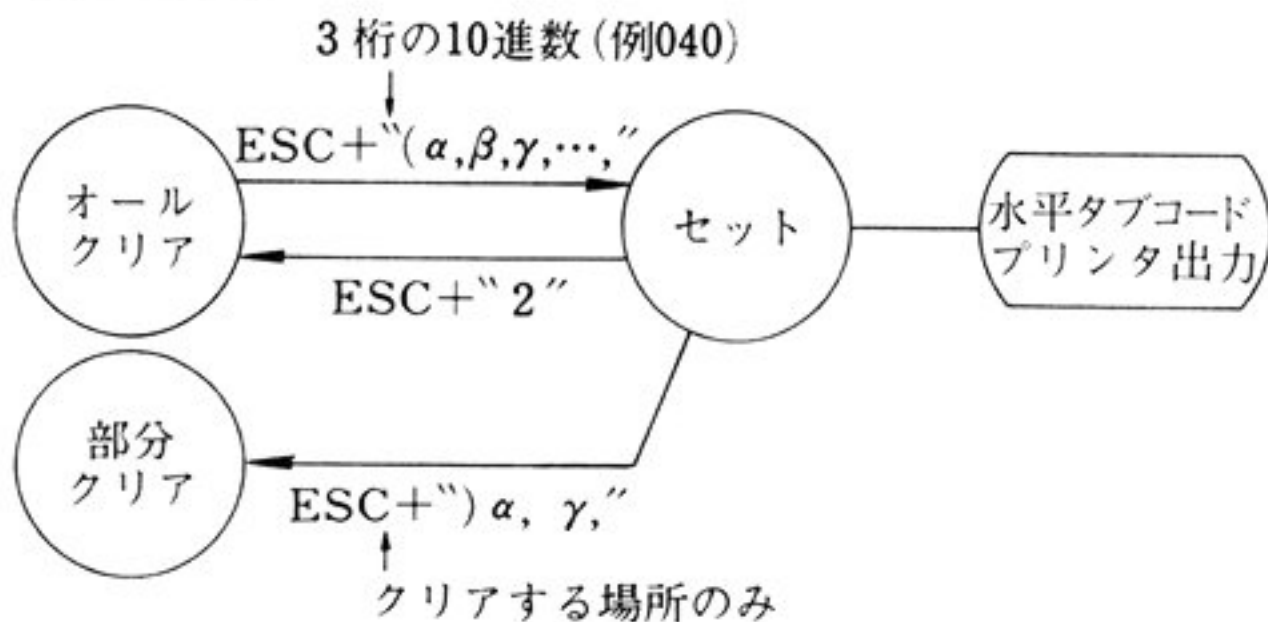
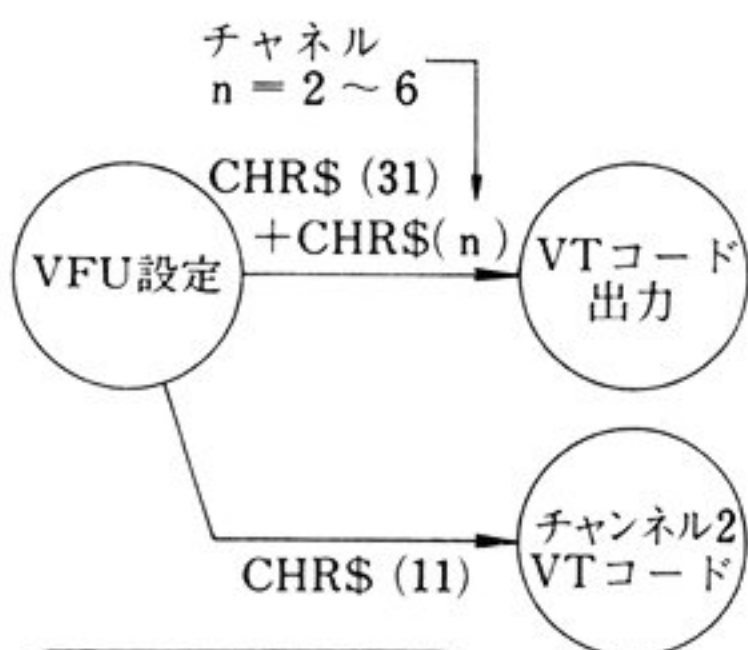
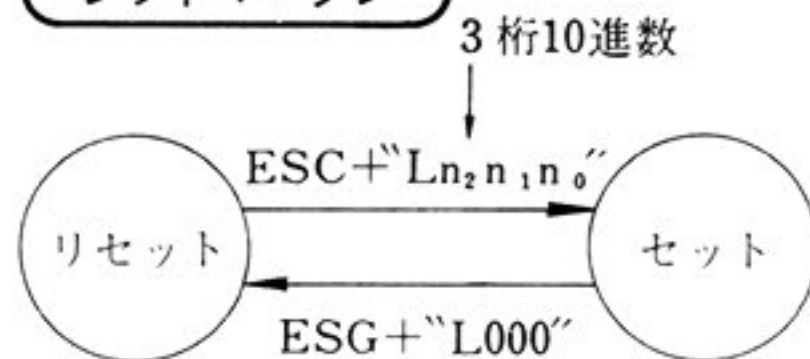
キャラクタコードの切換え

改行幅の切換え

n/144″　ESC+"T"+"n_1 n_0"
2桁10進数
(n＝1が半ドット分に相当)

n行改行　CHR$ (31)＋CHR$ (16＋n)
$1 \leq n \leq 15$

ビットイメージ

強調印字

改行方向切換

アンダーライン印字

DIPスイッチ機能

		SW1-1	SW1-2	SW1-3	SW1-4	SW1-5	SW1-6	SW1-7	SW1-8
	国別仕様切換				TOF間	DCI,3処理	1ラインフル	印字指令コード	CRの機能
		SW1-1	SW1-2	SW1-3					
ON	JA	×	×	×	72行	無効	LF有	CR LF VF FF	CR+LF
	US	×	○	×					
OFF	UK	○	○	×	66行	有効	LF無	CRのみ	CR
	GE	×	×	○					
	SW	○	×	○					

	SW2-1	SW2-2	SW2-3	SW2-4	SW2-5	SW2-6	SW2-7
	数字0表示	機器アドレス	機器アドレス設定		電源ON時印字モード	7〜8ビット切換	電源ON時セレクト・ディセレクト
			機器No	SW2-3			
ON		有効	0	×	×	7bit	セレクト
			1	○	×		
OFF	0	無効	2	×	○	パイカ	8bit	ディセレクト
			3	○	○			

プリンタの選択
ESC+"a"〜"d"
0 〜 3

セレクト解除
ESC+CHR$(96)

図4—4—1(b)

第5章

キー入力

第5章　キー入力

T-BASIC(Ver1.1)を対象に，プログラマブル・ファンクションキー，キー割込，リアルタイム・キースキャン等，キー入力についての解説し，合わせて，各入力命令の比較をしました．

5-1　プログラマブル・ファンクションキー

T-BASICでは，8個のファンクションキーを自由に定義して使うことができます．さらに，LABELキーで，ファンクションキーの内容を画面に表示することができます．また，初期のマニュアルには書かれていませんでしたが，KEY　ONでキーの内容が表示され，KEY　OFFで表示を消すことができます．

5-1-1　格納状態

電源投入時には，ファンクションキーは次のように設定されています．

```
key list

1  FILESm
2  LOAD "
3  SAVE "
4  ?TIME$m
5  EDIT .m
6  KEY
7  LIST
8  RUNm
Ok
```

ファンクションキーの内容は，メモリ上には，FD70～EDE0Hに格納されています．

```
FD70 : FF FF 46 49 4C 45 53 0D  00 00 00 00 00 00 00 00   ..FILES. ........ EB
FD80 : 00 00 4C 4F 41 44 20 22  00 00 00 00 00 00 00 00   ..LOAD." ........ DF
FD90 : 00 00 53 41 56 45 20 22  00 00 00 00 00 00 00 00   ..SAVE." ........ FE
FDA0 : 00 00 3F 54 49 4D 45 24  0D 00 00 00 00 00 00 00   ..?TIME$ ........ 3C
FDB0 : 00 00 45 44 49 54 20 2E  0D 00 00 00 00 00 00 00   ..EDIT.. ........ 2E
FDC0 : 00 00 4B 45 59 20 00 00  00 00 00 00 00 00 00 00   ..KEY... ........ C6
FDD0 : 00 00 4C 49 53 54 20 00  00 00 00 00 00 00 00 00   ..LIST.. ........ 29
FDE0 : 00 00 52 55 4E 0D 00 00  00 00 00 00 00 00 00 00   ..RUN... ;....... DF
```

これと同じものがT-BASICインタプリタにも書き込まれています．そのアドレスは，次の通りです．

T-BASIC（ROM版）Ver1.0：00F0～0160H
　　　　　　　　Ver1.1：00C2～0130H
　　　（DISK版）Ver1.1：00DD～013FH

5-1-2　ファンクションキーの定義

ファンクションキーは，ダイレクトモードで定義する方法の他に，プログラム中でも定義することができます．

例として，次のプログラムを挙げます．

```
10 '***
20 '*** PF SAMPLE 1
30 '***
100 KEY ON
110 KEY 1,"WIDTH 80"+CHR$(13)
120 KEY 2,"COLOR 7,0"+CHR$(13)
130 KEY 3,"LOAD"+CHR$(&H22)
140 KEY 4,"SAVE"+CHR$(&H22)
150 KEY 5,"LLIST "
160 KEY 6,"EDIT ."+CHR$(13)
170 KEY 7,CHR$(&H1E)+CHR$(3)+"LIST "+CHR$(5)
180 KEY 8,"RUN"+CHR$(13)
190 KEY LIST
200 END
```

ファンクションキーを定義するとき，Ver 1.1とDISK版では，文字変数を使うことができます．

文字変数を使うことができないVer 1.0では，メモリに直接書き込むことによって定義することにより，代用することができます．

次のプログラムは，ファンクションキーに，文字変数を使った例で，バージョンにかかわらず使用することができます．

```
100 '***
110 '*** PF SAMPLE 2
120 '***
130 KEY ON:WIDTH 80:DEFINT A-Z
140 PRINT "ｱｶﾀﾉ T-BASIC ﾊ ver 1.0 ﾃﾞｽｶ? (Y/N)";
150 INPUT A$
160 IF A$="Y" OR A$="y" THEN 270
170 '*
180 '* FOR VER 1.1 OR T-DISK
190 '*
200 INPUT" INPUT NUMBER OF PF KEY ( END = 0 ) :",KEYN
210 IF KEYN<0 OR KEYN>8 THEN 200
220 IF KEYN=0 THEN END
230 INPUT" INPUT FUNCTION FOR KEY :",KEYF$
240 KEY KEYN,KEYF$
250 KEY OFF:KEY ON
260 GOTO 200
270 '*
280 '* FOR VER 1.0
```

```
290 '*
300 KEYAD=&HFD72
310 INPUT" INPUT NUMBER OF PF KEY ( END = 0 ) :",KEYN
320 IF KEYN<0 OR KEYN>8 THEN 310
330 IF KEYN=0 THEN END
340 INPUT" INPUT FUNCTION FOR KEY :",KEYF$
350 KEYAD1=KEYAD+(KEYN-1)*16
360 FOR I=1 TO LEN(KEYF$)
370  POKE KEYAD1+I-1,ASC(MID$(KEYF$,I,1))
380 NEXT I
390 FOR I=LEN(KEYF$)+1 TO 16
400   POKE KEYAD1+I-1,0
410 NEXT I
420 KEY OFF:KEY ON
430 GOTO 310
```

このプログラムでは，Ver 1.0では，直接メモリに書き込むことによって，文字変数を使えるようにしています．

5-1-3　ファンクションキー割込

DISK版では，ファンクションキーを割込キーとして使うことができます．

ON KEY GOSUBで，ファンクションキー割込の処理サブルーチンを指定し，KEY(0) ONで割込を許可します．また，KEY(0) STOPで割込を保留し，KEY(0) OFFで禁止します．

この機能を利用して，ファンクションキーを鍵盤のように使ってみました．

```
10 '***
20 '***          KEY INTERRUPT SAMPLE
30 '***
100 WIDTH 36:SCREEN 1:CLS:COLOR 3
110 PRINT TAB(7);"*** ONEKEYBORD ***"
115 PRINT :COLOR 5
120 PRINT TAB(10);" PF1  :ﾄﾞ
130 PRINT TAB(10);" PF2  :ﾚ
140 PRINT TAB(10);" PF3  :ﾐ
150 PRINT TAB(10);" PF4  :ﾌｧ
160 PRINT TAB(10);" PF5  :ｿ
170 PRINT TAB(10);" PF6  :ﾗ
180 PRINT TAB(10);" PF7  :ｼ
190 PRINT TAB(10);" PF8  :ﾄﾞ
195 PRINT :COLOR 7
197 PRINT TAB(10);"END...[RETURN]"
200 ON KEY GOSUB 240,250,260,270,280,290,300,310
210 KEY (0) ON
220 IF INKEY$=CHR$(13) THEN KEY (0) OFF:END
230 GOTO 210
240 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 48,2:WEND:RETURN
250 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 50,2:WEND:RETURN
260 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 52,2:WEND:RETURN
270 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 53,2:WEND:RETURN
280 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 55,2:WEND:RETURN
290 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 57,2:WEND:RETURN
300 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 59,2:WEND:RETURN
310 KEY (0) STOP:WHILE INP(42)<>80 :SOUND 60,2:WEND:RETURN
```

このプログラムでは，左端のPF1からドレミファソラシドの音がするようになっています．

ファンクションキー割込は，このように機能を定義しておけばプログラム中でも割込で音を出す等の処理をすることが可能です．

5-2　キー入力

プログラム中でよく利用するキー入力命令としては，INPUT，INKEY$，INPUT$，LINE INPUT等があります．

5-2-1　INPUT

INPUTでは，とくに指定しない限り入力要求の〝?〟が出力されます．〝?〟を画面に残したくないときは，次のようにプロンプトのあとに〝,〟（カンマ）を付ければ？が出ません．

```
10 INPUT "A=",A
20 PRINT A
RUN
A=1.23
 1.23
Ok
```

また，数値変数の入力の際に数字以外の数を入力すると，?　Redo from start が表示され，２行下で再入力となります．

数値変数を入力する際に，39桁を越えた数を入力することはできません．また，文字変数では，254文字を越えた分は無視されます．次にあげるプログラムを実行してみて下さい（ただし，ROM版では，Overflowではなく，?OVとなります）．

```
10 INPUT "A=",A
20 PRINT A
30 INPUT "A$=";A$   :PRINT
40 PRINT A$
RUN
A=1.234567890123456789012345678901234567890
Overflow
?Redo from start
A=1
 1
A$=? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA
Ok
```

また，INPUT命令で〝,〟を入力したいときは，次のように入力する文を〝ダブルクォーテーション〟でくくれば可能です．

```
10 INPUT "A=",A
20 PRINT A
30 INPUT "A$=";A$   :PRINT
```

```
40 PRINT A$
RUN
A=1
 1
A$=? "A,B"

A,B
Ok
```

5-2-2　INPUT$

INPUT$は，入力する文字数が指定でき，また，カーソルコントロールキーの入力も可能です．メニューの入力の際などに便利な命令といえます．

また，エコーバックしませんが，それを利用して次のような使い方もできます．

```
10 '***
20 '***  INPUT$ SAMPLE
30 '***
100 CLS
110 LOCATE 10,0
120 PRINT"ﾊｲｹｲｼｮｸﾊ ﾅﾆｲﾛﾆ ｼﾏｽｶ  "
130 PRINT" Color code ｦ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ  :"
140 LOCATE 31,1:PRINT"       ";:LOCATE 31,1
150 A$=INPUT$(1)
160 IF A$="0" THEN PRINT" ｸﾛ   ";:GOTO 250
170 IF A$="1" THEN PRINT" ｱｵ   ";:GOTO 250
180 IF A$="2" THEN PRINT" ｱｶ   ";:GOTO 250
190 IF A$="3" THEN PRINT" ﾑﾗｻｷ ";:GOTO 250
200 IF A$="4" THEN PRINT" ﾐﾄﾞﾘ ";:GOTO 250
210 IF A$="5" THEN PRINT" ﾐｽﾞｲﾛ";:GOTO 250
220 IF A$="6" THEN PRINT" ｷｲﾛ  ";:GOTO 250
230 IF A$="7" THEN PRINT" ｼﾛ   ";:GOTO 250
240 GOTO 140
250 B$=INPUT$(1)
260 IF B$<>CHR$(13) THEN LOCATE 31,1:A$=B$:GOTO 160
270 COLOR ,VAL(A$)
280 END
```

5-2-3　INKEY$

INKEY$は，キーボードから1文字入力する関数ですが，実際には，先行入力がバッファにたまっているので，正確には〝キーボードバッファから1文字得る命令〞とした方がよいでしょう．ここで，先行入力について実験してみましょう．

```
10 FOR I=1 TO 10000:NEXT I
20 SOUND 40,20:FOR I=1 TO 1000:NEXT
30 A$=INKEY$:B$=B$+A$
40 IF A$<>"" THEN 30
50 PRINT B$,LEN(B$)
```

このプログラムを実行して，適当なキーを押し続け，音がしたら指をキーから離します．そうすると，キーバッファに何文字ためられるかを調べることができます．

実行すると，31文字までたまることがわかります．

5-2-4　入力命令の比較

以前にあげたキー入力命令の比較表を次に示します．

	INPUT	LINE INPUT	INKEY$	INPUT$
書　　式	INPUT A INPUT A$	LINE INPUT A$	A$＝INKEY$	A$＝INPUT$ (1)
入力指示の“?”	表示する	表示しない	表示しない	表示しない
プロンプト(“ ”)の出力	可	可	できない	できない
カーソルの表示	有	有	無	有
入力時のカーソル移動	有	有	無	不動
エコーバック	有	有	無	無
入力待ち	待つ	待つ	待たない	待つ
カンマの入力	カンマは変数の区切となる．	できる	できる	できる
コントロールコードの入力	不可	不可	可	可
入力文字数	数値変数：39桁以内 文字変数：254文字以内	254文字以内	1文字	254以下の指定した文字数
入力の終了	[RETURN] キー	[RETURN]キー	自動	自動
[RETURN] キーだけ押したとき	数値変数：0 文字変数：ヌルストリング	ヌルストリング	CHR$(13)	CHR$(13)
(STOP) キー [CTRL]─[C]	中断	中断	中断	中断
Break時のCONTによる再開	可	可	可	不可

図5－2－1　キー入力命令比較表

5-2-5　キーバッファのクリア

T-BASICのキーバッファは，FF01～FF20Hの31バイトとなっています．

キーが打たれたとき，いったんバッファにためて必要なときに(例えば，INPUT，INKEY$等)，バッファからとり出して使うようになっています．T-BASICでは，プログラム実行中でもバッファに打たれたキーのアスキーコードがためられるために，INPUTを実行したときに以前に打ったものが残っていて不快な思いをすることがあります．

このようなことを防ぐためには，キーバッファを0で埋めてしまえばよいのです．次のプログラムは，キーバッファの中身を調べるプログラムです．RUNしてSTOP以外のキーを押してみて下さい．キーバッファの内容が表示されます．止めたいときにはSTOPキーを押して下さい．

```
10 '***
20 '***      KEY BUFFER SUMPLE
30 '***
100 BUFFER=&HFF01
110 WIDTH 80:COLOR 7,0:KEY OFF:CLS
120 WHILE INKEY$<>"":WEND
130 FOR I=BUFFER TO BUFFER+30
140   PRINT CHR$(PEEK(I));
150 NEXT
160 PRINT:GOTO 120
```

キーバッファを0で埋めるとどうなるかを試してみましょう．

何もしない場合と，0で埋めた場合の両方を調べることにします．次のプログラムを実行して，比較してみましょう．

```
10 '***
20 '***          KEY BUFFER CLEAR SAMPLE
30 '***
100 BUFFER=&HFF01
110 WIDTH 80:COLOR 7,0:KEY OFF:CLS
120 PRINT"*** Clear ｼﾅｲ ﾄｷ ****"
130 PRINT:PRINT" ﾃｷﾄｳﾅ Key ｦ ｵｼﾃﾐﾖｳ"
140 FOR I=1 TO 10000:NEXT
150 PRINT" ｺﾚﾀﾞｹ ｺﾞﾐ ｶﾞ ﾀﾏﾘﾏｽ"
160 INPUT A$:PRINT
170 PRINT"*** Clear ｼﾀﾄｷﾊ  ****"
180 PRINT:PRINT" ﾃｷﾄｳﾅ Key ｦ ｵｼﾃﾐﾖｳ"
190 FOR I=1 TO 10000:NEXT
200 PRINT " ｺﾞﾐﾊ ﾀﾏﾘﾏｾﾝ"
210 FOR I=BUFFER TO BUFFER+31:POKE I,0:NEXT
220 INPUT A$
230 END
```

```
*** Clear ｼﾅｲ ﾄｷ ****

 ﾃｷﾄｳﾅ Key ｦ ｵｼﾃﾐﾖｳ
 ｺﾚﾀﾞｹ ｺﾞﾐ ｶﾞ ﾀﾏﾘﾏｽ
? LKJ;LKD;LKD;LJG:PMEMUFPIPUTJH

*** Clear ｼﾀﾄｷﾊ  ****

 ﾃｷﾄｳﾅ Key ｦ ｵｼﾃﾐﾖｳ
 ｺﾞﾐﾊ ﾀﾏﾘﾏｾﾝ
?
```

5-3　リアルタイムキースキャン

5-3-1　キーマトリクスを調べる

INKEY$では，キーバッファに先行入力がたまってしまうので，ゲーム等で要求されるリアルタイムキースキャンは困難です．

先行入力を除けばできないことはないのですが，速度が遅くなります．

次のプログラムは，INKEY$を使って，キャラクタ(Z)を動かす例です．

```
100 X=1:WIDTH 80:CLS
110 A$=INKEY$:IF A$="" THEN 140
120 LOCATE X-1,CSRLIN:PRINT "Z";
130 X=X+1:IF X=80 THEN PRINT:X=1
140 FOR I=1 TO 100:NEXT
150 GOTO 110
```

このように，INKEY$を使ってキー入力を行うと，キーを離してからもしばらくZが動きます．先行入力をとり除くため，60行に，

60 WHILE INKEY$ "" : WEND

を入れると先行入力は除かれますが，動きが不自然に遅くなります．

このように，INKEY$ではうまくキースキャンすることは困難です．

リアルタイムキースキャンには，機械語を使用するのがよいのですが，そのためにもキーボードスキャン信号と入力データについて説明しておきます．

パソピアでは，キーボードサーチ用にZ80 PIOが使用されています．

次の図表はキーボードスキャン信号と入力データの内容を示しています．

ポート	動作モード	端子	アクティブ	コントロール内容
A	出力モード3 (ビットコントロール) 割り込みなし 30H	A7	H	スピーカ発振を有効とします．
		A6	H	スキャンブロックCを有効とします．
		A5	H	スキャンブロックBを有効とします．
		A4	H	スキャンブロックAを有効とします．
		A3	H	各スキャンブロック内のそれぞれのスキャンラインを設定します．
		A2	H	
		A1	H	
		A0	H	
B	入力モード3 (ビットコントロール) LowレベルのOR条件により割り込みを発生する． 31H	B7	L	スキャン結果のデータ入力．
		B6	L	
		B5	L	
		B4	L	
		B3	L	
		B2	L	
		B1	L	
		B0	L	

図5―3―1　キーボードのコントロール内容(Z80PIO)

この図表のように，キースキャン信号はPIOのポートAのA_3～A_0から出力され，A_6～A_4によって各ブロックに振り分けられてマトリクスに12の端子に出力されます．

通常のキー打鍵の監視はすべてのスキャンラインをアクティブにして，ポートBのいずれかの端子がLOWレベルになって発生する割込により行われています．この割込が発生した後，各スキャンラインをスキャンしてキーコードが求められています．

たとえば，KSBラインをスキャンして，ビットデータがEFHなら，キーのMが押されたことが分ります．

ポートAに出力されるスキャン信号と，ポートBに入力するビットデータの関係は次の図表を見て下さい．

ポートA出力		スキャン信号	ポートB入力(ビット・データ)							
$A_6A_5A_4$	$A_3A_2A_1A_0$		B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0 0 1 (BLOCKA)	0 0 0 1	KS0			カナ	CTRL		CAPS LOCK	SHIFT	GRAPH
	0 0 1 0	KS1	7	6	5	4	3	2	1	0
	0 1 0 0	KS2	RETURN	.	CLR HOME	↑	↓	—	9	8
	1 0 0 0	KS3					TAB	DEL	ESC	→
0 1 0 (BLOCKB)	0 0 0 1	KS4	SPACE		←	INS	(STOP)	(COPY)	KANJI	LABEL
	0 0 1 0	KS5	PF8	PF7	PF6	PF5	PF4	PF3	PF2	PF1
	0 1 0 0	KS6	¥ ー	< ヘ	— ロ	Y ン	R ス	$ 4 ゥ ウ	0 ヲ ワ	! 1 ヌ
	1 0 0 0	KS7	[「 。	@ ゛	7 ヤ	U ナ	T カ	8 ュ ユ	3 ア	2 フ
1 0 0 (BLOCKC)	0 0 0 1	KS8	] 」 ム	* : ケ	) 9 ヨ	H ク	F ハ	& 6 ォ オ	% 5 ェ エ	= — ホ
	0 0 1 0	KS9	P ノ セ	O ラ	I ニ	J マ	G キ	E ィ イ	W テ	Q タ
	0 1 0 0	KSA	「 ; レ	L リ	K メ	N ミ	V ヒ	D ツ	S ト	A チ
	1 0 0 0	KSB	? / ・ メ	< , \ ル	< . ゜ ホ	M モ	B コ	C ソ	X ナ	Z ・ ッ

図5—3—2　スキャン信号とビットデータ

この図表を実際に確認するために，サンプル・プログラムを作ってみました．(STOP)以外のキーを押すと，押されたキーに対応するスキャンラインのビットが0になります(ただし，BASICなので反応は遅くなります)．ESCキーを押してしばらくすると止ります．

```
10 '***
20 '***    KEY SCAN    ( T-BASIC )
30 '***
100 SCREEN 1:WIDTH 80:CLS
110 COLOR 7,0:DEFINT A-Z
```

```
120 LOCATE 10,0:PRINT"[[  KEY SERCH  ]]
130 GOSUB 270
140 GOSUB 420
150 FOR I=0 TO &HB
160   OUT &H30,SCANLINE(I)
170   A=INP(&H31):A=INP(&H31) AND A
180   LOCATE 10,I+2:PRINT USING"###  ";A;
190   FOR J=7 TO 0 STEP -1
200     B.DAT=-((BIT(J) AND A)<>0)
205     PRINT B.DAT;
210   NEXT J
220   PRINT
230 NEXT I
240 A$=INKEY$
250 IF A$=CHR$(27) THEN OUT &H30,&H7F:END
260 GOTO 150
270 '
280 PRINT"               KS  7  6  5  4  3  2  1  0"
290 PRINT"    K S 0 :"
300 PRINT"    K S 1 :"
310 PRINT"    K S 2 :"
320 PRINT"    K S 3 :"
330 PRINT"    K S 4 :"
340 PRINT"    K S 5 :"
350 PRINT"    K S 6 :"
360 PRINT"    K S 7 :"
370 PRINT"    K S 8 :"
380 PRINT"    K S 9 :"
390 PRINT"    K S A :"
400 PRINT"    K S B :"
410 RETURN
420 '*
430 '* SCANLINE
440 '*
450 DIM SCANLINE(11),BIT(7)
460 FOR I=0 TO 2
470   HIGH=2^I
480   FOR J=0 TO 3
490     LOW=2^J
500     SCANLINE(I*4+J)=HIGH*16+LOW
510   NEXT J
520 NEXT I
530 FOR I=0 TO 7
540   BIT(I)=2^I
550 NEXT I
560 RETURN
```

5-3-2　機械語を使う

前述のとおり，リアルタイムでキー入力を使用するには，機械語を使うのが一番適当です．

そこで，機械語・キースキャン・ルーチンを作ってみました．

次のプログラムは，テンキーの2，4，6，8が押されているか調べ，それに応じて画面上の@(アットマーク)を動かすものです．

```
10 '***
20 '***   KEY SCAN ( MACHINE LANGUAGE )
30 '***
100 CLEAR ,&HEFFF
110 KBL$="3 ":KBD$="   "
120 SCAN=&HF000:IF SCAN<0 THEN SCAN=SCAN+2^16
130 KBA=SCAN+21
140 KBB=SCAN+22
150 '*
160 '* SET MEMORY
170 '*
```

```
180 POKE SCAN    ,&HF5                    :' PUSH    AF
190 POKE SCAN+1  ,&H3A                    :' LD      A,KBA
200 POKE SCAN+2  ,KBA-INT(KBA/256)*256 :'
210 POKE SCAN+3  ,INT(KBA/256)           :'
220 POKE SCAN+4  ,&HF3                    :' DI
230 POKE SCAN+5  ,&HD3                    :' OUT     30H
240 POKE SCAN+6  ,&H30                    :'
250 POKE SCAN+7  ,&HFB                    :' EI
260 POKE SCAN+8  ,&HDB                    :' IN      31H
270 POKE SCAN+9  ,&H31                    :'
280 POKE SCAN+10,&HEE                     :' XOR     0FFH
290 POKE SCAN+11,&HFF                     :'
300 POKE SCAN+12,&H32                     :' LD      KBB,A
310 POKE SCAN+13,KBB-INT(KBB/256)*256 :'
320 POKE SCAN+14,INT(KBB/256)            :'
330 POKE SCAN+15,&H3E                     :' LD      A,7FH
340 POKE SCAN+16,&H7F                     :'
350 POKE SCAN+17,&HD3                     :' OUT     30H
360 POKE SCAN+18,&H30                     :'
370 POKE SCAN+19,&HF1                     :' POP     AF
380 POKE SCAN+20,&HC9                     :' RET
390 '*
400 '*  START
410 '*
420 WIDTH 80:CLS:COLOR 7,0
430 X%=40:XX%=X%:Y%=12:YY%=Y%
440 LOCATE X%,Y%:PRINT"@";
450 POKE KBA,&H12
460 IF TIME-OVER=120 THEN 420
470 CALL SCAN                             :' SCAN LINE=1
480 IF PEEK(KBB)=&H4 THEN YY%=Y%+1:GOTO 540
490 IF PEEK(KBB)=&H10 THEN XX%=X%-1:GOTO 540
500 IF PEEK(KBB)=&H40 THEN XX%=X%+1:GOTO 540
510 POKE KBA,&H14 :CALL SCAN             :' SCAN LINE=2
520 IF PEEK(KBB)=&H1 THEN YY%=Y%-1:GOTO 540
530 GOTO 450
540 LOCATE X%,Y%:PRINT".";
550 IF XX%< 0 THEN 580
560 IF XX%>78 THEN 580
570 X%=XX%
580 IF YY%<0  THEN 440
590 IF YY%>24 THEN 440
600 Y%=YY%:GOTO 440
```

5-4　コントロールキー

5-4-1　コントロール・キャラクタ

コントロール・キャラクタはキャラクタコードの01Hから1FHまでのキャラクタで，この中には使用されていないものもあります．

プログラム中でコントロールキャラクタを使いたいときには，PRINT CHR$(n)とすれば使うことができます．例えば，PRINT CHR$(7)とすればベルが鳴ります．

コントロールキャラクタを見たいときにはファンクションキーに定義して，LABELキーでファンクションキーの表示を見ればよいでしょう．

5-4-2　コントロール・コード一覧

コントロール・コードとシンボルとキー，機能の対応は次のようになっています．

16進	10進	シンボル	シンボルの意味	対応するキー	機能
00	0		null	CTRL + @	スペース
01	1	SH	Start of Heading(ヘッディング開始)	〃 + A	
02	2	SX	Start of Text(テキスト開始)	〃 + B	カーソルを1項目ごとに左へ移す.
03	3	EX	End of Text(テキスト終了)	〃 + C	STOPと同意(プログラムの実行を中止して, BASICコマンドモードに戻る)
04	4	ET	End of Transmission(伝送終了)	〃 + D	
05	5	EQ	Enquiry(問合わせ)	〃 + E	カーソル位置から後ろをまっ消する.
06	6	AK	Acknowledge(肯定応答)	〃 + F	カーソルを1項目ごと右に移す.
07	7	BL	Bell(ベル, ブザー)	〃 + G	内蔵のブザーを鳴らす.
08	8	BS	Back Space(後退)	〃 + H	DEL カーソルのすぐ左の1文字を削除する.
09	9	HT	Horizontal Tabulation(水平タイプ)	〃 + I	8文字ごとの水平タブ
0A	10	LF	Line Feed(改行)	〃 + J	カーソルを次の行へ移す.
0B	11	HM	Home(VT) Vertical Tabulation(垂直タブ)	〃 + K	カーソルをホームポジション画面左上に戻す.
0C	12	CL	Clear(FF) Form Feed(改頁)	〃 + L	画面を消去してカーソルをホームポジションに戻す.
0D	13	CR	Carriage Return(復帰)	〃 + M	カーソルを次の行の先頭に移す.
0E	14	SO	Sift-out(シフトアウト)	〃 + N	カーソルを1項目ごと右へ移す.
0F	15	SI	Sift-in(シフトイン)	〃 + O	画面の表示を無効にする. 再度押すと再開する.
10	16	DE	Data Link Escape(伝送制御拡張)	〃 + P	
11	17	D1	Device Control 1 (装置制御1)	〃 + Q	
12	18	D2	Device Control 2 (装置制御2)	〃 + R	INS カーソルの位置から右側を1文字分右へずらす.
13	19	D3	Device Control 3 (装置制御3)	〃 + S	
14	20	D4	Device Control 4 (装置制御4)	〃 + T	
15	21	NK	Negative Acknowledge(否定応答)	〃 + U	
16	22	SN	Synchronous idle(同期信号)	〃 + V	
17	23	EB	End of Transmission Block(送伝ブロック終了)	〃 + W	
18	24	CN	Cancel(取消し)	〃 + X	
19	25	EM	End of Medium(媒体終端)	〃 + Y	
1A	26	SB	Substitute (文字置換)	〃 + Z	
1B	27	EC	Escape(拡張)	ESC	実行の1時中断
1C	28	→	(FS) File Separtor(ファイル分離)	→	カーソルを1つ右へ移す.
1D	29	←	(GS) Group Separator(グループ分離)	←	カーソルを1つ左へ移す.
1E	30	↑	(RS) Record Separator(レコード分離)	↑	カーソルを上の行へ移す.
1F	31	↓	(US) Unit Separator(ユニット分離)	↓	カーソルを下の行へ移す.

図5—4—1 コントロールコード一覧

第6章

漢字入出力

第6章　漢字入出力

6-1　ディスクを使う

6-1-1　漢字パターンと漢字ファイル

漢字1字は16×16ドットのドットパターンで構成されています．このパターンのデータは2バイト×16の32バイトデータになっています．次の図に示すように，2バイト×16の中に漢字1字が表示される形を示し，数字はバイト位置を示します．また，各1ビットが1ドットに対応しています．

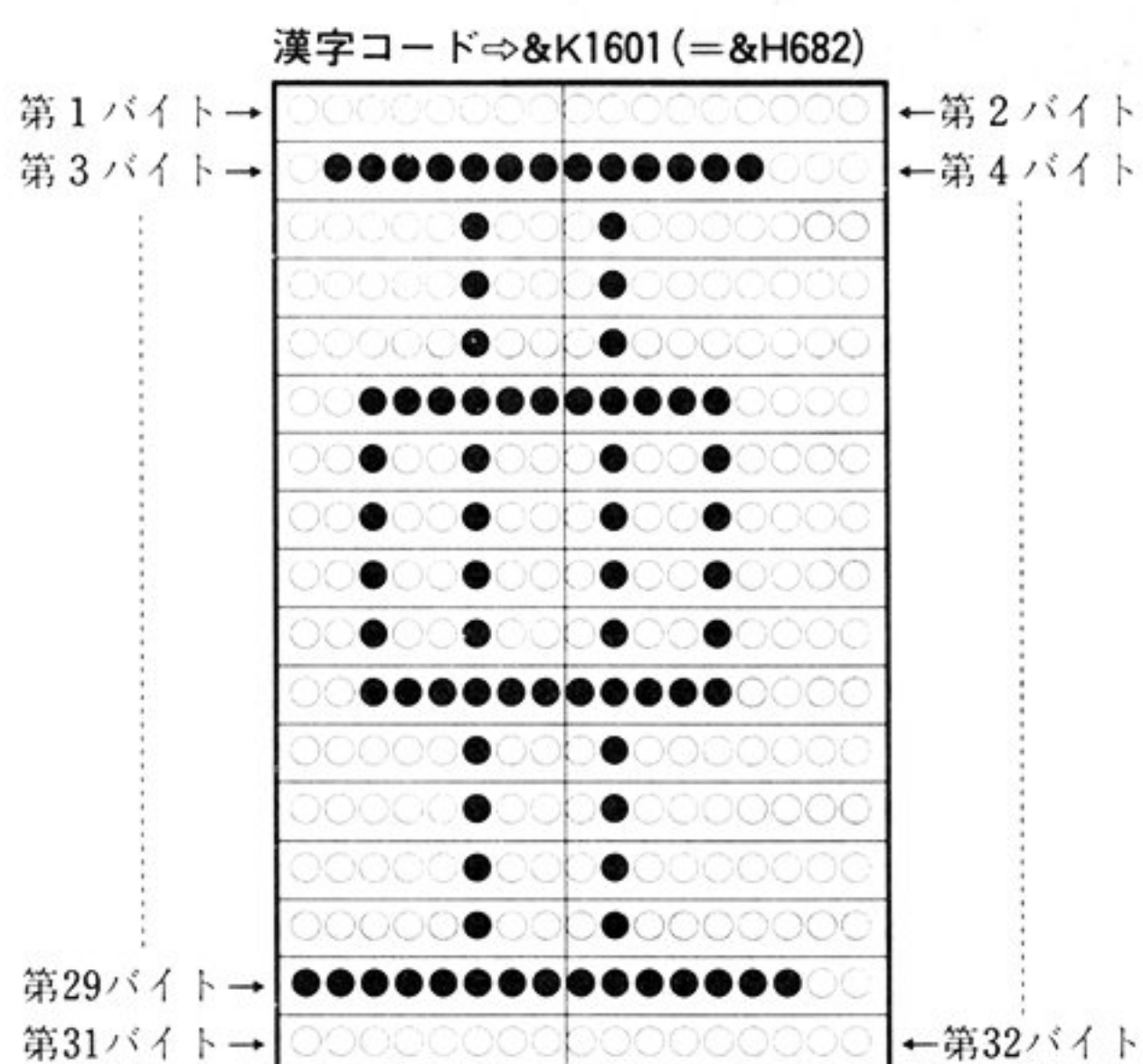



図6―1―1　漢字フォントのフォーマット

漢字パターンのファイルは，T-DISKBASICのシステムディスクの19トラックから34トラックを占め，その中に3666字分のパターンが格納されています．

アドレッシングは次の図のようになっています．また，途中の未定義部分（8区1点～15区19点，48区1点～83区94点，87区21点～94区94点）は，指定してもエラーは出ませんが空白を出力するだ

けです.

図6－1－2 漢字パターンファイル

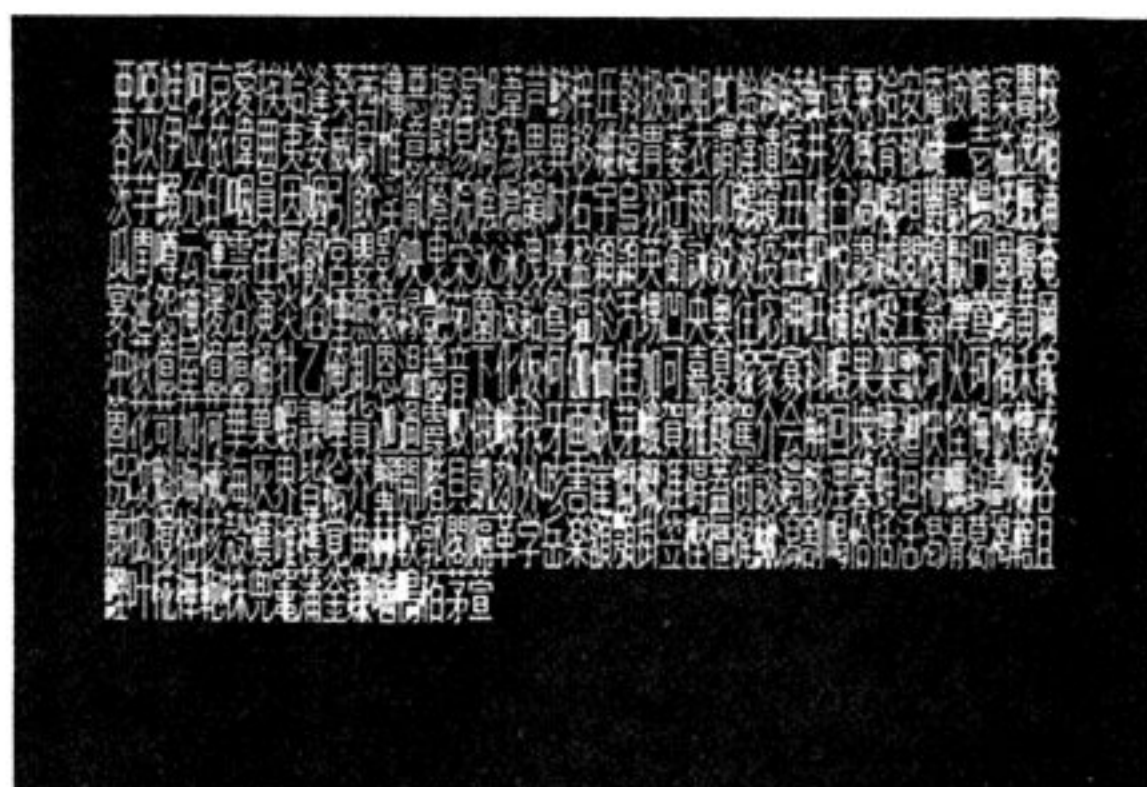



写真6－1
漢字の出力例

6－1－2 漢字コードの内部表現

漢字を選択する方法としては次の2つの方法があります.

例 「亜」を出力する ・数値で表す.

```
PUT@(10, 10), KANJI(1666)
```

・漢字コード(&K, &Jをつけて)表す.

```
PUT@(10, 10), KANJI(&K1601)
```

いずれの方法でも, インタプリタ内では数値として扱われています. 漢字コードの内部値は, 256～9091となります.

$$\text{内部値}=\left\{\begin{array}{l}(\text{区番号}-1)*94+(\text{点番号}-1)\\(\text{区コード}-33)*94+(\text{点コード}-33)\end{array}\right\}+256$$

$$=\left\{\begin{array}{l}\text{区番号}*94+\text{点番号}+161\\\text{区コード}*94+\text{点コード}-2879\end{array}\right\}\qquad(256\sim9091)$$

(例) • 1区1点(&K0101)…… 256 (＝94＋1＋161)
• 16区1点(&J 3021)……1666 (＝48＊94＋33－2879)

10　C＝1666
20　PUT@(10, 10) KANJI (C), PSET
これは(10, 10)の位置に16区1点の"亜"を表示する.

図6－1－3　漢字コードの内部表現

6-1-3　ディスクからの入力

ディスクから漢字パターンを入力するには，漢字パターンの内部値から，ディスク上の格納アドレスを求め，目的のドットパターンを含む1セクタを入力することによって可能です．そのためには，内部値を1区1点を0とする漢字パターン内の相対座標に変換する必要があります．

図6－1－4　漢字ファイルのアドレス

この図は内部値と相対位置の関係を示しています．また，次のプログラムは実際にDISK$で漢字パターンをディスクから読んで画面に出力するものです．

```
1000 '***
1010 '*** KANJI READ FROM DISK
1015 '***
1017 SCREEN 1
1020 FIELD #0,128 AS C1$,128 AS C2$
1030 DATA 2[illegible]  256,658,1410,752,3666,7050,3384,3968,32767
1040 FOR I=1 TO 10
1050   READ T(I)
1060 NEXT I
1065 CLS:PRINT "ｵｵﾓｼﾞﾃﾞ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ"
1066 '***** INPUT KANJI CODE
1070 INPUT "Jis or Kuten or End  ( J/K/E ):",B$
1071 IF B$="E" THEN END
1075 IF B$="J" THEN B$="&J":GOTO 1090
1080 IF B$="K" THEN B$="&K" ELSE GOTO 1070
```

```
1090 INPUT "KANJI CODE:",A :A=VAL(B$+STR$(A))
1100 I=1:B=A
1110 IF B<T(I) THEN PRINT "ILLEGALL":GOTO 1070
1120 I=I+1:B=B-T(I):I=I+1
1130 IF B<T(I) THEN 1160
1140 I=I+1
1150 GOTO 1110
1160 PRINT USING"ﾅｲﾌﾞﾁ =####     ｿｳﾀｲｲﾁ =###";A;B
1170 S=INT(B/8)
1180 O=(B-8*S)*32+1
1190 S=S+19*32+16
1200 H=INT(S/16)
1210 S=S-16*H+1
1220 T=INT(H/2)
1230 H=H-2*T
1235 PRINT USING "ｷﾛｸｼﾃｲﾙ ﾄﾗｯｸ=>## ﾍｯﾄﾞ=>## ｾｸﾀ=>##";T;H;S
1240 PRINT          "FONT ｶｲｼ bit =>";O
1250 D$=DSKI$(1,H,T,S)           :' READ DISK
1260 IF O>128 THEN D$=C2$:O=O-128
1270 P$=MID$(D$,O,32)
1280 P=VARPTR(P$):IF P<0 THEN P=P+2^16
1290 N=PEEK(P+1)+PEEK(P+2)*256
1300 INPUT "ﾋｮｳｼﾞ ｲﾁ  X,Y ";X,Y
1310 FOR I=0 TO 30 STEP 2
1320   PRESET (X,Y+I/2)
1330   PX=PEEK(N+I)   :GOSUB 1370
1340   PX=PEEK(N+I+1):GOSUB 1370
1350 NEXT I
1360 GOTO 1070
1370 '***** PSET SUB
1375 FOR J=7 TO 0 STEP -1
1380   IF (PX ¥ (2^J))=1 THEN PSET STEP(1,0) ELSE PRESET STEP(1,0)
1390   PX=PX MOD (2^J)
1400 NEXT J
1410 RETURN
```

6-1-4　漢字データをディスクから消去

漢字データはディスク内で60クラスタ占めていて，システムディスクのフリーエリアは40クラスタしか残っていません．ユーティリティのVOLCOPYを使うとシステムとともに漢字データもCOPYされてしまいます．システムディスク全てに漢字データをのせておく必要はないので，ディスクから漢字データを消去したいときには次のプログラムを実行して下さい．なお，このプログラムは，FATを書き換えて漢字データで予約されている領域をユーザーに解放するものです．

```
10 '***
20 '***  KANJI DATA ERASE FROM DISK
30 '***           ( T-DISK BASIC )
100 WIDTH 80:COLOR 7,0:SCREEN 0:CLS
110 PRINT TAB(20);"<<<  KANJI DATA ERASER >>>"
120 PRINT
130 PRINT TAB(10);"Mount DRIVE 1 for KANJI System disk"
140 PRINT TAB(10);"       ................ Ok ->  hit RETURN key"
145 W$=INKEY$:IF W$<>CHR$(13) THEN 145
150 PRINT
160 BUFFER=&H96CF
170 '
180 '     CHECK FD1 HAS KANJI DATA
190 '
200 FIELD#0,139 AS A$,117 AS B$
210 DUMMY$=DSKI$(1,0,18,14)
220 FOR I=75 TO 139
230   IF PEEK(BUFFER+I)<>&HFE THEN STATUS=STATUS+1
240 NEXT I
250 IF STATUS>3  THEN PRINT CHR$(7); "This disk has not kanji-data" :END
```

```
260 '
270 '      WRITE FAT 0FFH
280 '
290 FOR I=80 TO 139
300   POKE BUFFER+I,&HFF
310 NEXT I
330 FOR I=14 TO 15
340   DSKO$ 1,0,18,I
350 NEXT I
360 END
```

6-2　漢字ROMPAC2

T-BASIC ver1.1およびT-DISKBASICでは漢字パターンが格納されている漢字ROMPACを使用することができます．この漢字ROMPACの使用方法はBASICのバージョンによって異っています．

・T-BASIC　ver 1.1の場合

機械語サブルーチンをメモリにロードし，コールします．

```
NEW
Ok
CLEAR ,&HF450
Ok
BLOAD #-1,"TKANJI"
Found:TKANJI
Ok
I=&HF450:CALL I
Ok
PUT@(50,80),KANJI(1666)
Ok      亜
```

使用法は，PUT＠(X，Y)，KANJI(漢字コード)として使います．(X，Yはスクリーン上の座標)なお，このサブルーチンをLOADしてあればGET＠，PUT＠も使用できます．また，フリーエリアは約300バイト減少します．

・T-DISK BASIC　ver 1.0の場合

サブルーチンをLOADしてコールします．次に示す手順で行います．

```
NEW
Ok
CLEAR ,&HFDC0
Ok
BLOAD #-1,"TDKANJI"
Found:TDKANJI
Ok
I=&HFDC0:CALL I
Ok
PUT@(50,80),KANJI(1666)
Ok      亜
```

また，外字はディスクから読み出されます．その他はT-BASIC ver 1.1と同様です．

6-3　直接プリンタに出力する

次のプログラムは，漢字をディスクから直接プリンタに出力するものです．この際の問題点は，漢字データの1バイトは横方向8ドットであるのに対し，プリンタに出力するデータの1バイトは縦方向8ドットなので縦横を変換する必要があるということです．この変換は機械語サブルーチンで行っています．機械語サブルーチンの説明はリストの注釈を見て下さい．

```
1000 '***
1010 '*** KANJI OUTPUT PRINTER
1020 '***
1030 CLEAR ,&HEFFF
1040 CLS
1050 DIM T(10)
1060 '*MACHINE LANGUAGE SUBROUTINE
1070 ' sub1:::::::::::::::::: control
1080 DATA D5,23,5E,23,56,EB,E3,23,5E,23,56,E1,CD,14,F0,23,CD,14,F0,C9
1090 ' sub2:::::::::::::::::: 8 bit ﾀﾃ-ﾖｺ ﾍﾝｶﾝ
1100 DATA E5,0E,08,06,08,E1,E5,CB,06,1F,23,23,10,F9,12,13,0D,20,F0,E1,C9
1110 ' sub3:::::::::::::::::: printer output
1120 DATA 46,23,5E,23,56,1A,13,CD,F4,FF,10,F9,C9
1130 ADR=&HF000                  :' ﾀﾃ-ﾖｺ ﾍﾝｶﾝ SUB
1140 LPTOUT=&HF029               :' PRINTER OUTPUT
1150 FOR I=0 TO 53
1160   READ A$: POKE ADR+I,VAL("&h"+A$)
1170 NEXT I
1180 '-----------------------------------------------------------------------
1190 FIELD #0,128 AS C1$,128 AS C2$
1200 DATA 256,256,658,1410,752,3666,7050,3384,3968,32767
1210 FOR I=1 TO 10: READ T(I) : NEXT I
1220 C$=CHR$(&H1B)+"S0016"     :' dot mateix output code
1230 LF1$=CHR$(&H1B)+"T17"     :' GRAPHIC FEED
1240 LF2$=CHR$(&H1B)+"A"       :' NORMAL FEED
1250 CRLF$=CHR$(&HD)+CHR$(&HA) :' CR/ LF
1260 '-----------------------------------------------------------------------
1270 INPUT "Kuten or END ( k/e ):",B$
1280 IF B$="E" OR B$="e" THEN END
1290 IF B$="k" OR B$="K" THEN B$="&K" ELSE 1270
1300 PRINT "KANJI code";:X=POS(0):A$=""
1310 LOCATE X,CSRLIN:PRINT A$;"    ";:I$=INPUT$(1)
1320 IF I$=CHR$(13) THEN 1330 ELSE IF I$=CHR$(&H1D) THEN A$=LEFT$(A$,LEN(A$)-1):
GOTO 1310 ELSE A$=A$+I$
1330 IF LEN(A$)<>4 THEN 1310 ELSE LOCATE X,CSRLIN:PRINT A$:A=VAL(B$+A$)
1340 I=1: B=A
1350 IF B<T(I) THEN PRINT "Illegal": GOTO 1270
1360 I=I+1: B=B-T(I): I=I+1
1370 IF B<T(I) THEN 1410
1380 I=I+1
1390 GOTO 1350
1400 '-----------------------------------------------------------------------
1410 PRINT "ﾅｲﾌﾞﾁ ";A;" ｿｳﾀｲ ｲﾁ ";B
1420 '
1430 S=INT(B/8)
1440 O=(B-8*S)*32+1              :' CHARACTER POSITION
1450 S=S+19*32+16: H=INT(S/16)
1460 S=S-16*H+1                  :' SECTOR ADDRESS
1470 T=INT(H/2)                  :' TRACK ADDRESS
1480 H=H-2*T                     :' HEAD
1490 PRINT "TRACK :";T;" HEAD :";H;" SECTOR :";S;" POSITION :";O
1500 '
1510 D$=DSKI$(1,H,T,S)           :' READ PATTERN
1520 IF O>128 THEN D$=C2$:O=O-128
1530 P$=MID$(D$,O,32)            :' GET ONE PATTERN
1540 '-----------------------------------------------------------------------
1550 UP$=STRING$(16,0): LP$=STRING$(16,0)
1560 AP$=LEFT$(P$,16) : CALL ADR(AP$,UP$)
1570 AP$=RIGHT$(P$,16): CALL ADR(AP$,LP$)
1580 CALL LPTOUT(C$)   : CALL LPTOUT(UP$): CALL LPTOUT(LF1$): CALL LPTOUT(CRLF$)
1590 CALL LPTOUT(C$)   : CALL LPTOUT(LP$): CALL LPTOUT(LF2$): CALL LPTOUT(CRLF$)
1600 GOTO 1270
```

第7章

OA-BASICの内部構造

第7章　OA-BASICの内部構造

OA-BASICは東芝が独自に開発したBASICで，東芝のオフコンBP-100のBASICをパソピア用に改良したものであり，BP-100とプログラムの互換性がほぼ保たれています．ファイル処理機能が優れていることや，BCD演算であることから誤差が少ないという特徴を持っているので，文字どおりOAに対しての強力なツールとして使うことができます．

この章ではOA-BASICの構造とファイル処理についてスポットをあてて説明します．

7-1　メモリ内部の状態

7-1-1　メモリ・マップ

OA-BASICはROM版のものとDISK版のもとではメモリの使用状態が異っています．

DISK-BASICのメモリの使用状態を説明します．DISK-BASICではROMの32Kバイトと RAMの64Kバイトを使用して動いています．ほとんどのコマンド・ステートメントを実行するときにはRAMモードで，また直接実行命令を実行するときにはROMモードで，というようにROMとRAMをバンク切換しながら動いています．（1-1-3参照）

DISK-BASIC起動時のメモリ・マップは図7-1-1のようになります．図中のポインタ等のシンボルについては後述します．

この図を見ると，ROM領域のCLOAD等の書き込まれている部分のRAMは，データエリアと拡張エリアになっています．このうちデータエリアは，文字列のデータの格納などに使用され，拡張エリアは，BASICの拡張命令の処理ルーチンを格納するために使用されます．

中間言語プログラムエリアには，BASICプログラムが中間言語で格納され，変数エリアには変数名などが格納されます．BASICインタープリタのワークエリアは，入出力のバッファや各種のポインタの格納に使用されています．

ROM版では，電源投入時にROMの内容がRAMにコピーされ，ROMモードで動作する一部のコマンド（LOADなど）の実行を除けばRAMモードで動作します．ROMで使用されるコマンドはCLOAD, CSAVE, TERM等，BASICプログラムの実行時には使用されることのない命令で，この命令が入力されたときには，バンクを切り換えてROMモードにして実行します．

データエリア2はデータエリア1と同様に使われます．

以下にROM版のメモリ・マップを示します.

OA-BASIC
インタプリタ

RENUM/CLOAD
CSAVE/CLOAD
DSAVE/TERM

未使用

OATOP
TOPPR
BTMPR
BTMVR
VATOP
OCTOP
VATOP

領域	アドレス
	0000
OA-DISK-BASIC	
	JKBUF (6983)
漢字バッファエリア	
	DATOP (6FF0)
ディスクファイル制御情報エリア (FCB)	
未使用	
	FMEML (7FBF)
拡張文用エリア	
	LEMEM (8000)
中間言語プログラムエリア	
変数名登録エリア	
データ・エリア	
未使用	
配列テーブルエリア	
変数データアドレス登録エリア	
	HLMEH (F049)
DISK-BASIC用ワークエリア	
	(F245)
BASICインタプリタワーク・エリア (入出力バッファ等)	
	(FF02)
スタックエリア	
	(FFFF)

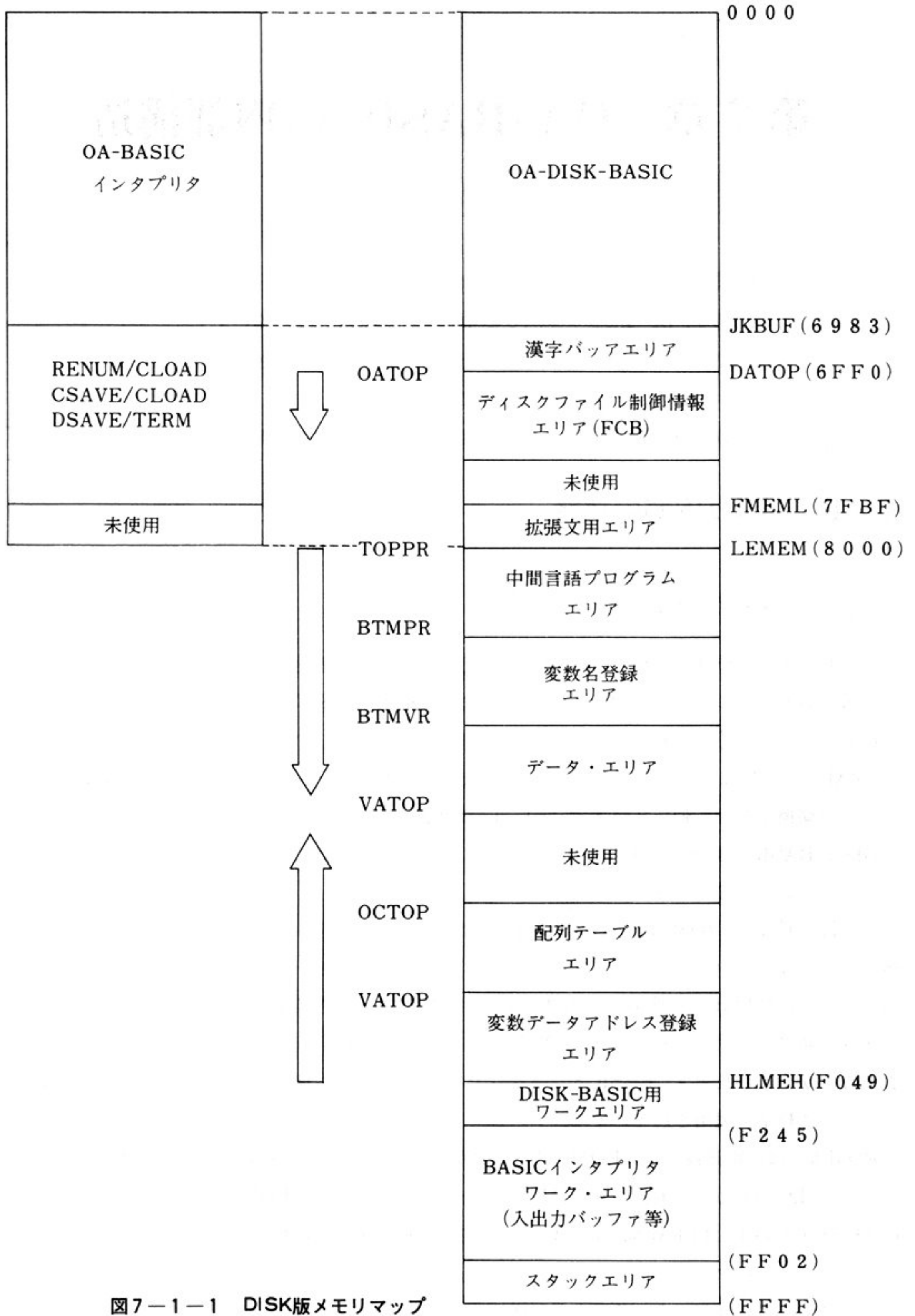



図7−1−1　DISK版メモリマップ

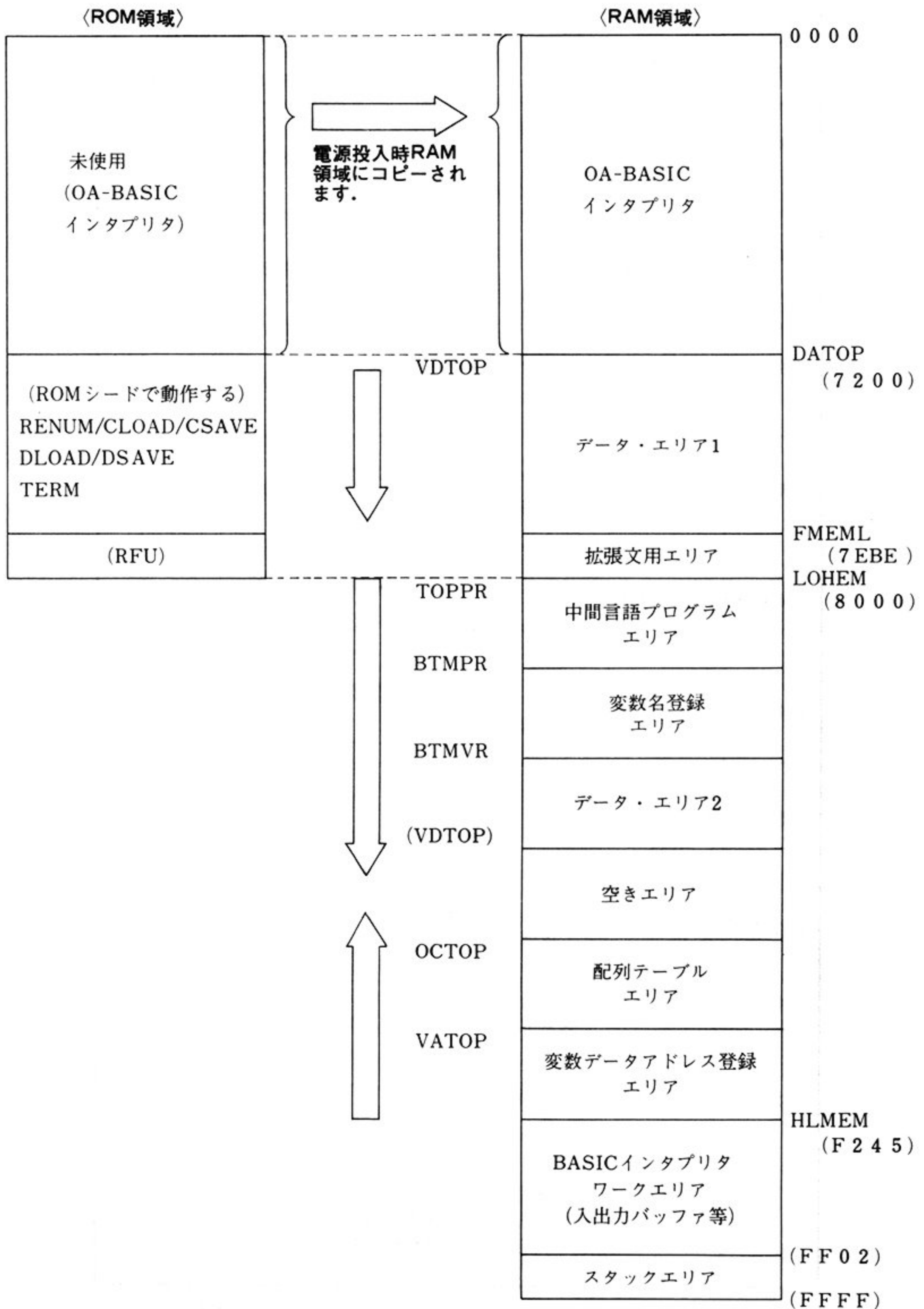



図7－1－2　ROM版メモリ・マップ

ここで，メモリ・マップで使用したポインタやその他のポインタについて簡単に説明します．

アドレス (16進数)	シンボル名	内容	初期値(16進数) ROM版	初期値(16進数) DISK版	可変
F428 29	LOMEM	利用者領域の先頭アドレスが格納される．（LOMEMコマンドを実行すると変化する．）	8000	8000	○
F42A 2B	HIMEM	利用者領域の最終アドレスが格納されている．	F245	F049	固定
F405 6	BTMPR	中間言語プログラムの最終アドレスが格納されている．	8000	8000	○
F407	TOPPR	中間言語プログラムの先頭アドレスが格納されている．	8001	8001	○
F409	VATOP	変数アドレスエリアの現在の先頭アドレスが格納されている．	F244	F048	○
F40B	VDTOP	データ・エリアの現在の先頭アドレスが格納されている．	7200	8020	○
F835	OCTOP	配列テーブルエリアの現在の先頭アドレスが格納されている．	F244	F048	○
FABC	BTMVR	変数名エリアの現在の最終アドレスが格納されている．	8000	8000	○
F40F	CURPT	中間言語プログラムのカレントアドレスポインターが格納されている．	8001	8001	○
F838	DATOP	〈ROM版〉 データ・エリアの先頭アドレスが格納されている． 〈DISK版〉 ディスクファイル制御情報エリアの現在のアドレスが格納されている．	7200 (固定)	6FF0	

図表7－1－3　メモリ・マップで使用したシンボルの意味

BASICインタプリタにとって必要な数値(例えばプログラムの格納領域の先頭アドレス)は，ポインタと呼ばれるワークアドレスに格納されています．次の表で示されるアドレスと，それに続くアドレスの2バイトに分けて記憶されます．

たとえば，シンボルLOMEMのポインタ，F428Hに00Hが，F429Hに80Hが書き込まれていたとするなら，ユーザ領域の先頭アドレスは8000Hである，ということになります．なお，表中の〝可変″の項はそのポインタの値を変化させることの可否を示します．

また，入出力バッファは次のように割り当てられています．

先頭アドレス(16進数)	バイト数	内　　容
FCFF	(バイナリー)	ライン入力バッファ上に入力されている，データバイト数を示す．
FB00	256	ライン入力バッファ 入力された1行分のコマンドまたはソースプログラムのバッファあるいは，INPUT/LINPUT集のバッファとして使用される．
F48B	2 (バイナリー)	ファイル入出力バッファ上のデータバイト数を示す．
F48D	256	ファイル入力バッファ フロッピー，カセット，データファイル，プリンタファイルの入出力バッファとして使用される．
6983	480	漢字編集バッファとして使用される．
FED8	2	RS232C用バッファの先頭アドレスが入っている．

図表7－1－4　入出力バッファ一覧

7-1-2　プログラム領域

ユーザ領域の先頭アドレスはLOMEMのポインタにより示され，電源投入時には8000Hとなっています．このアドレスはLOMEMコマンドによって変更することができます．中間言語プログラムは，ユーザ領域の先頭アドレスの次のアドレスより始まり，シンボルBTMPRのポインタで示されるアドレスまで格納されています．

実際のプログラムの格納状態は，次のプログラムを使えば調べることができます．打ち込む前にLOMEM8000を実行して下さい．

```
0010REM
0020REM    Ascii dump for OA-BASIC
0030REM
0040%OPEN #"pr1","adump",1,"O"
0050 DIM P$%79(0):
     WIDTH 80,25
```

```
0060 INPUT "start address:";SAD$
0070 INPUT " end  address:";EAD$
0080 HEXA$=SAD$
0090 GOSUB 320
0100 SADH=INT(HEXA/16)
0110 HEXA$=EAD$
0120 GOSUB 320
0130 EADH=INT(HEXA/16)
0140*
0150*
0160 FOR I=SADH TO EADH
0170   IF INT(I/8)=I/8 THEN PRINT :* PRINT #1;" "
0180   P$(0)=HEX$(I*16)+" : "
0190   FOR J=0 TO 15
0200     M=PEEK(I*16+J)
0210     P$(0)=P$(0)+HEX$(M)+" "
0220    IF (J=7)+(J=15) THEN P$(0)=P$(0)+" "
0230   NEXT J
0240   FOR J=0 TO 15
0250     M=PEEK(I*16+J)
0260    IF (M<128)*(M>31) THEN P$(0)=P$(0)+CHR$(M) ELSE P$(0)=P$(0)+"."
0270    IF J=7 THEN P$(0)=P$(0)+" "
0280   NEXT J
0290   PRINT P$(0):*PRINT #1;P$(0)
0300 NEXT I
0310 END
0320*
0330* hexa code to 10
0340*
0350 HEXA=0
0360 FOR I=1 TO LEN(HEXA$)
0370   A=ASC(MID$(HEXA$,I,1))
0380   IF (A)=$30)*(A<=$39) THEN HEXA=HEXA*16+A-$30:
       GOTO 400
0390   IF (A)64)*(A<71) THEN HEXA=HEXA*16+A-55:
       GOTO 400
0400 NEXT I
0410 RETURN
```

（注意および説明）

プリンタに出力したいときは40行，170行，290行の＊(アスタリスク)をとって下さい．

320行からのサブルーチンで16進から10進に変換しています．

また，入力は大文字で行って下さい．

このプログラムを用いれば，プログラム領域をアスキーダンプすることができます．このプログラムを使ってこのプログラムの格納されている領域をダンプすると次のようになります．

```
start address:8000
 end  address:807F
8000 : EF 10 00 07 00 BE 52 45  4D A0 EF 20 00 22 00 BE   ......RE M.. ."..
8010 : 52 45 4D 20 20 20 20 41  73 63 69 69 20 64 75 6D   REM    A scii dum
8020 : 70 20 66 6F 72 20 4F 41  2D 42 41 53 49 43 A0 EF   p for OA -BASIC..
8030 : 30 00 07 00 BE 52 45 4D  A0 EF 40 00 1E 00 BE 2A   0....REM ..@....*
8040 : 4F 50 45 4E 20 23 22 70  72 31 22 2C 22 61 64 75   OPEN #"p r1","adu
8050 : 6D 70 22 2C 31 2C 22 4F  22 A0 EF 50 00 16 01 D4   mp",1,"O "..P....
8060 : 01 E7 FE 79 00 EB FE 00  00 EA EE 9A FE 80 00 ED   ...y.... ........
8070 : FE 25 00 EF 60 00 16 01  C9 FA 22 73 74 61 72 74   .%..`... .."start
```


プログラムをダンプしてみると，マイクロソフト系のT-BASIC等と次の点が異っていることに気づくと思います．

1)行の区切りを示すのがEFHである．

2)リンカは次の行の先頭アドレスではなく，次のEFまでのバイト数を示している．

3)変数は名称ではなく，何番目の変数かという番号で書き込まれる．

4)REM文の中間コードを示すBEHの後に〝REM″や〝*″や〝,″がアスキーコードで格納されている．すなわち，REM文では＊や，を使った方がメモリ効率がよい．

7-1-3　変数の格納状態

OA-BASICは，カナ変数を使えまた変数は40文字まで識別することができます．変数の処理はマイクロソフト系のBASICとは大幅に異っています．

プログラムの後に変数のテーブルがあり，プログラム中には何番目の変数かということを示す1バイトの数が書き込まれています．変数のデータは前述のデータエリアに書き込まれています．

変数データのテーブルには，変数名の文字数1バイトと変数名が書き込まれ，ここには変数格納のポインタはありません．

変数の格納アドレスは，シンボルVATOPで示されるアドレスから変数名テーブルの順に格納され，変数の型と格納アドレスが示されています．変数の格納の状態を次の図に示します．

図7－1－5　変数の格納

変数名テーブル

プログラムの直後にあり，次のように書込まれる．

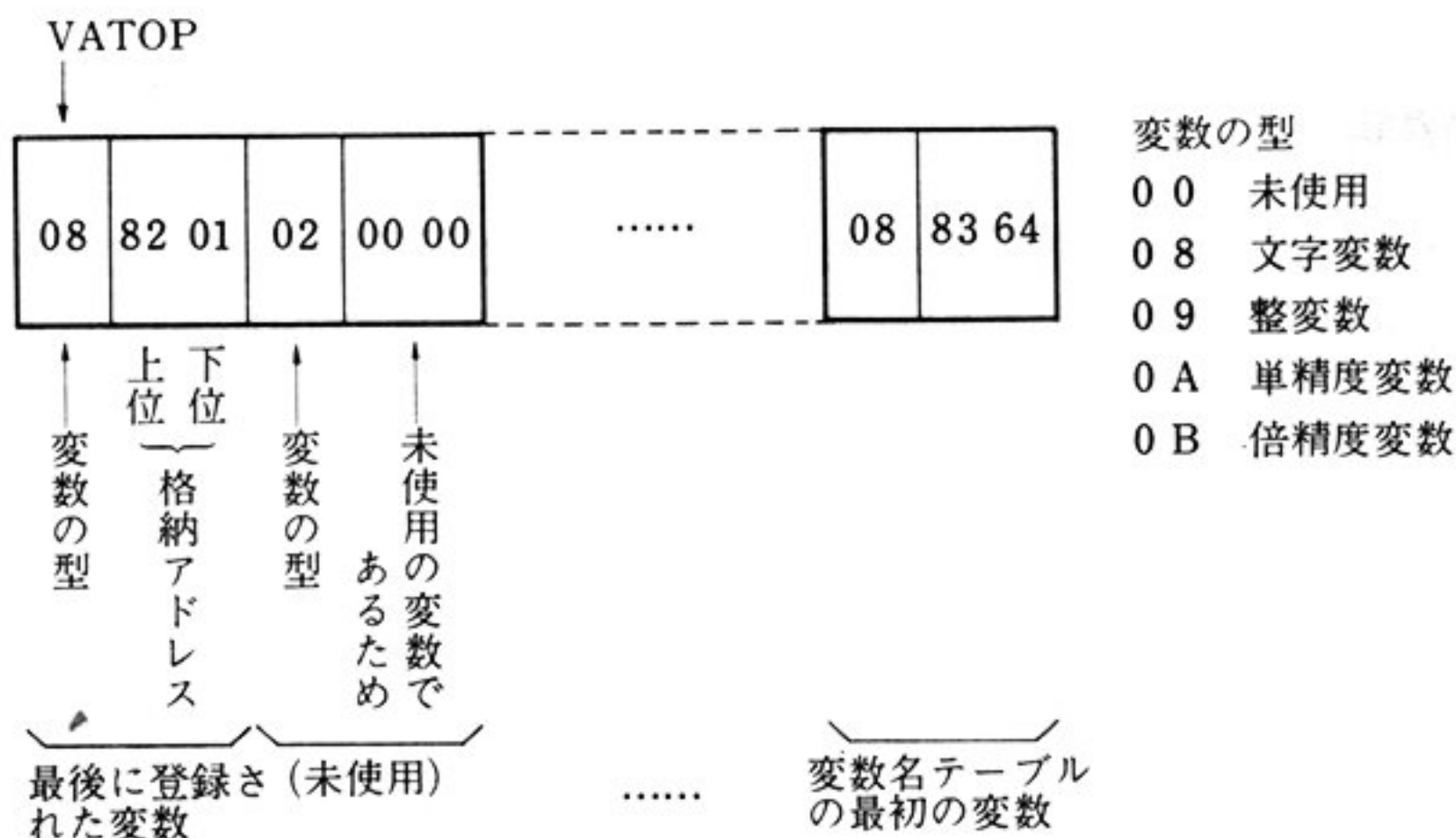
例)

```
8310 : 01 BD EF 03 53 41 44 01  49 00 02 50 24 04 53 41  ....SAD. I..P$.SA
8320 : 44 24 04 45 41 44 24 05  48 45 58 41 24 04 53 41  D$.EAD$. HEXA$.SA
8330 : 44 48 04 48 45 58 41 04  45 41 44 48 01 4A 01 4D  DH.HEXA. EADH.J.M
8340 : 01 41 41 01 41 00 00 00  00 00 00 00 00 00 00 00  .AA.A... ........
```

メモリこのテーブルにおける変数の順番(変数番号)が書込まれる．

例えば，SADなら００，Ｉなら０１となる．

例　変数番号　nの度数の格納アドレスは次のように求める。

- F409・F40AHを調べ変数アドレスエリアの先頭アドレスを調べる．〔ADR1＝PEEK($F409)＋PEEK$($F40A)〕　⇓
- 変数アドレスエリアには次のように書込れているので変数の格納されているアドレスを知ることができる。

格納形式　　　　　ADR(A%)はこのアドレスを示す．

⇩　　　⇦2桁ずつ区切り上位，下位を入れ替える．

整変数　A%＝1234⇨ | 34 | 12 |

単精度　A　＝.12345E－14⇨ | 2C | 56 | 34 12 |

3AHに指数部を加えたもの．S3A＋(－14)＝S2C

倍精度　A#＝.1346789D＋21⇨ | 4C | 90 78 56 34 01 |

注）プログラム入力時に123E＋12などと打ってもリストでは.123E＋15と変化します．

文字変数　AS＝ABC⇨ | 03 | 41 42 43 |

文字数　　　アスキーコードで格納される．

7-1-4　識別コード

識別コードは変数の前に付けられ，そのコードに続く数値の型を示します．例えば，A＝1234という文はメモリ内では次のように表現されます．

01DEFE3412　(01は変数の格納された順番を示す)

DEは〝＝〞を示す中間言語で，FEは整数を示す識別コードです．また，数値すべてBCD形式(2進化10進数)で格納されるので，1234が3412と格納されています．これは，単精度，倍精度ともに同様です．

識別コードは次のようになっています．

FA　文字列
FC　倍精度数
FD　単精度数
FE　整数

7-1-5　中間言語

プログラム中の各命令は，メモリ節約のためと実行速度向上のため，中間言語形式でメモリに格納されています．OA-BASICの中間言語は次のようになっています．

図表７－１－６(a)　OA-BASICの中間コード表

上位 / 下位	9	A	B	C	D	E	F	F90	F91	F93	F94
0		BSAVE	ELSE	RANDOMIZE	FOR	< >	* *	FNA	FNQ	POS	ERR
1		BLOAD	EOF	READ	COPY	< =	∧	FNB	FNR	FKEY	FLAG
2		MOTOR	USING	PRINT	ERASE	= <		FNC	FNS	SQR	FRE
3		AT	PURGE	POKE	END	> =		FND	FNT	ATN	PI
4		TPUT	CLOSE	PUT	DIM	= >		FNE	FNU	COS	POINT
5		TGET	OPEN	WRITE	DEF	#		FNF	FNV	EXP	PSET
6		BEEP	BUILD	ON		/		FNG	FNW	FIX	PRESET
7		SFLG	TROFF	NEXT		*	$(16進用)	FNH	FNX	LOG	ERL
8		WAIT	TRON	LET	CALL	−		FNI	FNY	SIN	KANJI&
9		COMMON	SEARCH	INPUT		+	関数(F9**)	FNJ	FNZ	TAN	KUTEN$
A	WIDTH		CLEAR	IF	STEP	)	ストリング	FNK	ABS	ASC	CIN$
B	SCREEN	INPUT	OUT	UPDATE	TO	(		FNL	CDBL	CHR$	
C	PRESET		STOP	GOTO	>	;	倍精度	FNM	CSNG	HEX$	
D	PSET		RETURN	CHAIN	<	,	単精度	FNN	CINT	SPACE$	
E	LINE	MID$	REM	GET	=	:	整数	FNO	INT	SPC	
F	COLOR	RENAME	RESTORE	GOSUB	> <	行の終り	プログラムの終り	FNP	SGN	STRING$	

注）　FA～FEHは識別コード

図表７－１－６(b)　OA-BASICの中間コード

KEYWORD	TOKEN				
#	E5	FIX	F936	NEXT	C7
$	F7	FKEY	F931	NULL	F928
(	EB	FLAG	F941	ON	C6
)	EA	FNA	F900	OPEN	B5
*	E7	FNB	F901	OUT	BB
**	F0	FNC	F902	PEEK	F929
+	E9	FND	F903	PI	F943
,	ED	FNE	F904	POINT	F944
-	E8	FNF	F905	POKE	C3
/	E6	FNG	F906	POS	F930
:	EE	FNH	F907	PRESET	9C
;	EC	FNI	F908	PRESET	F946
<	DD	FNJ	F909	PRINT	C2
<=	E1	FNK	F90A	PSET	9D
<>	E0	FNL	F90B	PSET	F945
=	DE	FNM	F90C	PURGE	B3
=<	E2	FNN	F90D	PUT	C4
=>	E4	FNO	F90E	RANDOMIZE	C0
>	DC	FNP	F90F	READ	C1
><	DF	FNQ	F910	REM	BE
>=	E3	FNR	F911	RENAME	AF
ABS	F91A	FNS	F912	RESTORE	BF
ADR	F924	FNT	F913	RETURN	BD
ASC	F93A	FNU	F914	RIGHT$	F923
AT	A3	FNV	F915	RND	F92A
ATN	F933	FNW	F916	SCREEN	9B
BEEP	A6	FNX	F917	SEARCH	B9
BLOAD	A1	FNY	F918	SFLG	A7
BSAVE	A0	FNZ	F919	SGN	F91F
BUILD	B6	FOR	D0	SIN	F938
CALL	D8	FRE	F942	SPACE$	F93D
CDBL	F91B	GET	CE	SPC	F93E

CHAIN	CD	GOSUB	CF	SQR	F932
CHR$	F93B	GOTO	CC	STEP	DA
CIN$	F94A	HEX$	F93C	STOP	BC
CINT	F91D	IF	CA	STR$	F92E
CLEAR	BA	IN#	F925	STRING$	F93F
CLOSE	B4	INPUT	AB	TAB	F92B
COLOR	9F	INPUT	C9	TAN	F939
COMMON	A9	INT	F91E	TGET	A5
COPY	D1	IRND	F926	TIM	F92C
COS	F934	KANJI&	F948	TO	DB
CSNG	F91C	KIN$	F927	TPUT	A4
DEF	D5	KUTEN$	F949	TROFF	B7
DIM	D4	LEFT$	F920	TRON	B8
ELSE	B0	LEN	F921	UPDATE	CB
END	D3	LET	C8	USING	B2
EOF	B1	LINE	9E	VAL	F92F
ERASE	D2	LOG	F937	WAIT	A8
ERL	F947	MID$	AE	WID	F92D
ERR	F940	MID$	F922	WIDTH	9A
EXP	F935	MOTOR	A2	WRITE	C5

7-2　ディスク・ファイル

7-2-1　ディレクトリ

データおよびプログラムの外部記憶装置として，ミニフロッピィディスクを使用することができます．

両面倍密のフロッピィディスクは，次のようなフォーマットで使用されています．

図7－2－1　フォーマット

このディスクフォーマットのうち，データセット・ラベルにはディスクに格納されているファイルの名称や格納位置などのデータが書き込まれています．このデータセット・ラベルは次のようなフォーマットで書き込まれています．

位置	1–4	5	6–13	14	15–20	21	22	23–27	28	29–33	34	35–39	40
内容	HDHI		ファイル名	ISF用	パスワード名	ISF用	ISF用	ブロック長		BOE		EOE	
n(m)	4 (A)	1	B (A)	1 (B)	6 (A)	1 (B)	1 (B)	5 (A)	1	5 (A)	1	5 (A)	1

位置	41	42	43	44	45	46–47	48–53	54–57	58	59–60	61–62	63–64	65–74	75–79	80
内容		アクセス許可	ライト・プロテクト	BASICフラグ	BUTフラグ	アドレスブロック内レコード		レコード長	ブロッキングファイル	収容レコード数 (SF)	Idle Block Pointer (ISF)	ファイル編成		EOD	
n(m)	1	1 (A)	1 (A)	1 (A)	1 (A)	2 (B)	6 (A)	4 (A)	1 (B)	2 (B)	2 (B)	2 (A)	10 (A)	5 (A)	1

フロッピーディスク上に定義されたファイルに対して作成される．

- 1ファイル：128バイト
  81～128バイトは、ダミー
- 1メディアに対し41ファイルまで定義可能
  0トラック　SIDE 0　セクタ8～16（1セクタ128バイト）
  0トラック　SIDE 1　セクタ1～16（1セクタ256バイト）

（注）フォーマット

内容
n(m)

n：バイト数
m：タイプ
A＝アスキーコードイメージ
B＝バイナリイメージ

図7－2－2　データセットラベルのフォーマット

この各部分はそれぞれ次のような意味をもっています．

図表7－2－3　データセットラベルの内容

カラム位置	タイプ	値	内　　容
1－4	アスキー	HDR 1	データセットラベルであることを示す．(固定)
6－13	アスキー	8文字以内	ファイル名を示す．
14	バイナリ		ISF ファイルのインデックスレベルを示す．
		00H	初期状態(データ未出力)
		02H	主索引/副索引1レベル
		03H	主索引/副索引1/副索引2レベル
15－20	アスキー	6文字以内	パスワード名を示す．全て空白の時は，パスワード無しと判断する．
21	バイナリ	1～14	ISF ファイルのキー長を示す．
22	バイナリ	1～255	ISF ファイルのキーの先頭カラム位置(キーポジション)を示す．
23～27	アスキー	␣␣256	データブロック長を示す．(␣␣256固定)
29－33	アスキー	TTHSS	ファイルの先頭トラックセクタを示す．(BOF)
			TT：トラック(0～39) H：ヘッド(0 or 1) SS：セクタ(1～16)
35～39	アスキー	TTHSS	ファイルの最終トラックセクタを示す．(EOF)
42	アスキー		アクセス許可を示す．
		空白	アクセス可能であることを示す．
		空白以外	アクセス不可能であることを示す．
43	アスキー		ライトプロテクト
		空白	書き込み可能であることを示す．
		空白以外	書き込み不可能であることを示す．(PUT#　WRITE#文)
44	アスキー		ASIC フラグを示す．
		B	BASIC 専用ファイルであることを示す．

カラム位置	タイプ	値	内　　　　容
45		B以外	アウトプットフラグであることを示す.
		O	アペンドが可能なファイルであることを示すRF, SVファイルに1件以上のデータが書き込まれている時,“0”となる.
		空白	アペンドが不可能なファイルであることを示す. ○ISファイルは全て空白 ○ファイル又スペースが確保されているだけで, データが1件を書き込まれていない状態の時空白になる. (OPEN＃文の入出力属性にAを指定することはできない.)
46〜47	バイナリ	0〜511	ブロック内レコードアドレス(SF, RFの時有効)アペンドモード(追加)でオープンされた時, レコードを追加するブロック内のバイト位置を示す.
54〜57	アスキー	1〜9999	レコード長を示す. RF/ISFファイルに対してだけ有効.
58	バイナリ		ブロッキングファクターを示す.
		2 nn	nnは, 1レコードのブロック数を示す. (レコード長＞ブロック長)
		0 nn	nnは, 1ブロック中のレコード数を示す. (レコード長≦ブロック長)
59−60	バイナリ	0〜9999	現在作成されているレコード数を示す. (SVファイル時に有効)
61〜62	バイナリ		Idele Blocke Pointer (ISFのみ有効) 空きブロックのチェーンセクターを示す. B6Eを1とした相対セクター器号
63〜64	アスキー	2桁	ファイル編成を示す.
		SV	順編成ファイルであることを示す.
		RF	乱編成ファイルであることを示す.
		LS	索引順編成ファイルであることを示す.
75〜79	アスキー	5桁	データ収容の最終セクター一つのトラックセクタを示す.

ミニフロッピィディスクに関するルーチンおよびポインタは次のようになっています．

図7－2－4　ディスク関係の処理ルーチンとポインタ

アドレス (16進数)	値 (16進数)	内　　容	使　用　例
0533		ミニフロッピールーチンのアドレスでCALLすることにより，前もって設定されたパラメータに従い，種々の処理を行う．	MFDD＝$D533 〜 CALL　MFDD
FEAD	(1バイト)	ミニフロッピーに対する処理動作を指示する． バイト・アドレス 〈機　能〉	CADR＝$FEAD 〜 POKE CADR, $86
	86	①シークアンド・リード(86H) 指定のトラック/ヘッド/セクターからデータを入力し，そのデータを入出力バッファに格納．	(シークアンド・リード) その他動作コマンドには，次のものがある． ○8DH：Seek & Write
	C5	②シークアンドライトチェック(C5H) 入出力バッファの内容を指定のトラック/ヘッド/セクターに書き込みます．またその後，書き込まれたデータを再度読み込みチェックを行います． (入出力バッファ内は，変わらない．)	○07H：Return to Zero ○04H：Sense Drive Status ○06H：Read Data ○07H：Write Data ○0FH：Seek ○0DH：Write ID
	85	③シークアンドライト(85H) 入出力バッファの内容を指定のトラック/ヘッド/セクターに書き込む．	通常の処理においては，上記のものは使用しない．
FEAC	(1バイト)	処理を行うFDDの装置番号を設定する． アドレス	DADR＝$FEAC 〜 POKE DADR, $01
	00	FDD NO.1の装置	(FDD NO2の装置を処理の対象とする)
	01	FDD NO.2の装置	
	02	FDD NO.3の装置	
	03	FDD NO.4の装置	

アドレス (16進数)	値 (16進数)	内　　容	使用例
FEAD FEAE	(2バイト)	入力出バッファのアドレスを設定する． アドレス MSS　　　LSS Low ⋮ High	入力出バッファは，DIM文で設定する変数領域を利用すると便利
		アドレス設定は，Low/Highを反転し指定する．	DIM IOBUF$＊ 255(1)
		(例)入出力バッファをC000Hとする時には，次のように設定する． FEAD　0 ⋮ 0 FEAE　C ⋮ 0	≀ BADR=ADR(IOB UF$(0)) ≀ POKE $FEAD, BADR INT(BADR/256) ＊256 POKE $FEAE, INT(BADR/256)
FEAF FEB0	(2バイト)	入出力データのデータ長を設定する．	256バイトの入出力
		データ長から1を引いたバイナリーデータを設定する	BSZ=256−1 ≀
		入出力バッファのアドレス設定と同じでメモリバイトのLow/Highを反転させ設定する．	POKE $FEAF, BSZ−INT(BSZ/ 256)＊256 POKE $FEBO, INT(BSZ/256) ≀
FEB1	(1バイト) 00〜22	入出力を行うトラック番号を設定 (0トラックから35トラック)	トラック2，ヘッド0，セクター9を設定 TTHSS=$FEB1 ≀
FEB2	(1バイト)	入出力を行うヘッド(サイド)番号を設定．	TT=2：M=0： SS=9 ≀
	0 0	表サイド	POKE TTHSS, INT(TT) POKE TTHSS+1 INT(H)
	0 1	裏サイド	POKE TTHSS+2 INT(SS)

アドレス (16進数)	値 (16進数)	内　　容	使用例
FEB3	(1バイト) 01～10	入出力を行うセクタ番号を設定 (1セクターから16セクター)	
FE34	(1バイト)	エラー発生時のリトライ回数をバイナリーで設定. BASICインタープリタ内部では10回(OAH)	RADR＝FEB4 ≀ POKE RADR, $OA
FE3C	(1バイト)	ミニフロッピー入出力ルーチンが設定する終了ステータスが格納される(Read only) 内容はコード	IOSTS＝$FE3C ≀ CALL $0533
	38("8")	正常終了	IF PEEK(IOSTS)
	30("0")	FDD not Ready(ドアオープン等)	＜＞"8" TH
	32("2")	IDフィードエラー	THEN
	33("3")	Seekエラーおよび欠トラックの検出	GOTO nnnn
	34("4")	CRCエラー	
	35("5")	Write CheckエラーおよびFile Unsafe エラー	
	36("6")	Deleted Sector検出	
	37("7")	入力パラメーターエラー	
	39("9")	Deleted SectorでCRCエラー	
	47("G")	Write Protectエラー	

このうち，$0533Hはディスクの I/Oルーチンで，このルーチンは次のようにすれば利用できます．この場合は各ポインタにアクセスするドライブ等のデータをPOKEなどで書き込んでからルーチンをコールするのですが，この場合トラック・セクタ等の数値がおかしければディスクの内容が破壊されることがあるので注意が必要です．(図7-2-5参照)

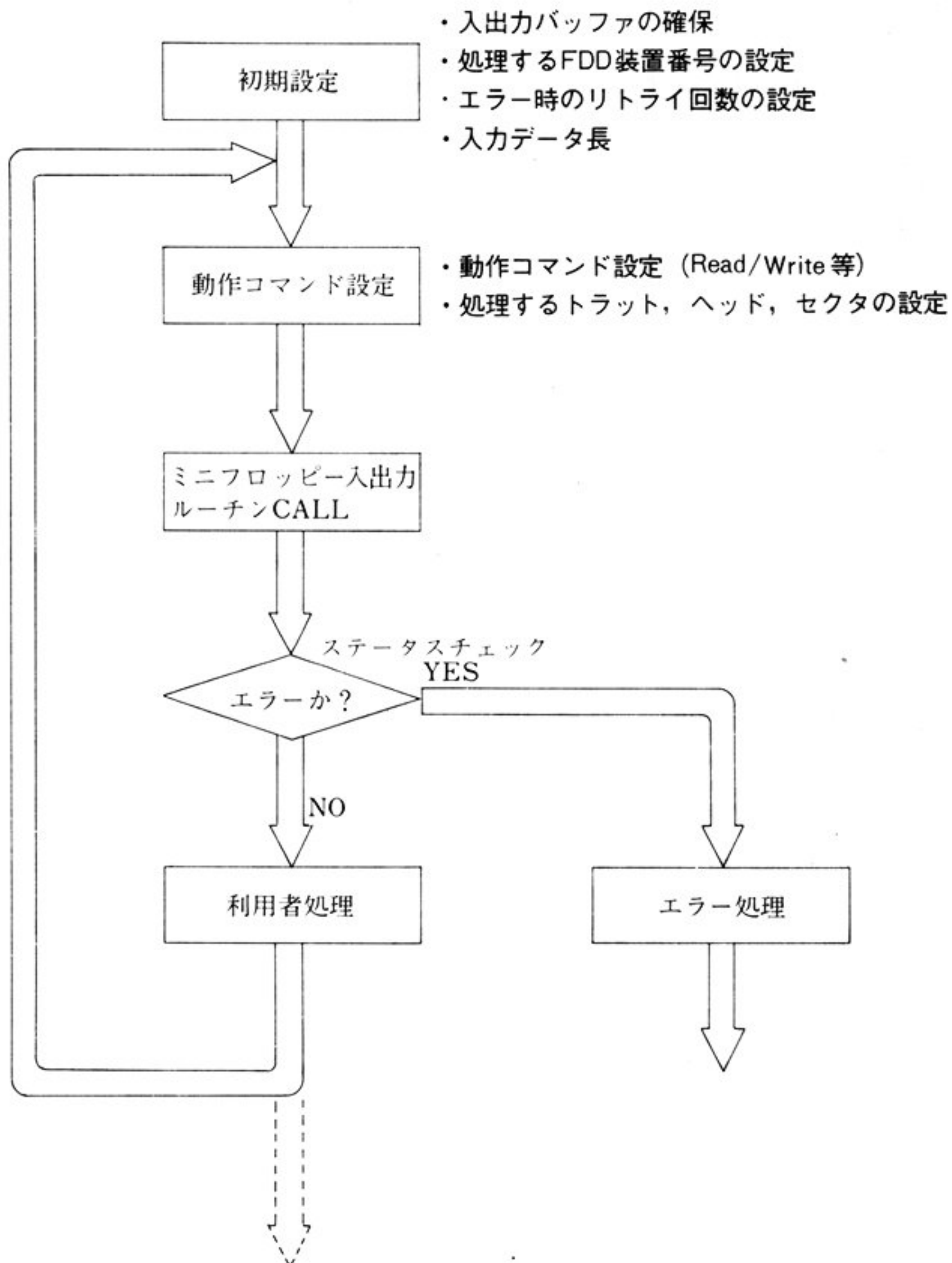



図7―2―5　ミニフロッピー入出力ルーチンの使用法

7-2-2　ミニフロッピィディスクのダンプ

ディスクのダンプを行いたい場合は，OA-BASICではヘッド・トラック・セクタを指定してディスクの内容をメモリにコピーさせるような命令がありません．そのような作業をしたいときには，前項で説明した内部サブルーチンを使う必要があります．次に示すプログラムは$0533の内部ルーチンを使用しています．

```
0010 ***
0020 ***     FDDUMP for OA-BASIC
0030 ***
0040 CLEAR
0050 DIM FDATA$*255(1)
```

```
0060 WIDTH 80,25:
     SCREEN 0
0070 ERASE
0080 PRINT TAB(15,2),"***** Flloppy disk dump program *****"
0090 PRINT TAB(10,5);:
     INPUT " Do you use printer? (y/n)";IO$
0100 IF (IO$="y")+(IO$="Y") THEN OPEN #"PR1","PRINTER",1,"O":
     IO=1 ELSE IO=0
0110 PRINT TAB(10,9);:
     INPUT "DRIVE  :";DRNO
0120 IF (DRNO>4)+(DRNO<0) THEN PRINT TAB(10);"ILLEGAL":
     GOTO 110
0130 PRINT TAB(10,10);:
     INPUT "TRACK  :";TRACK
0140 IF (TRACK>35)+(TRACK<0) THEN PRINT TAB(10);"ILLEGAL":
     GOTO 130
0150 PRINT TAB(10,11);:
     INPUT "HED    :";HED
0160 IF (HED=1)+(HED=0)=0 THEN PRINT "ILLEGAL":
     GOTO 150
0170 PRINT TAB(10,12);:
     INPUT "SECTOR NUMBER FROM ,TO :";SCTST,SCTEND
0180 IF (SCTST<1)+(SCTST>16)+(SCTEND<1)+(SCTEND>16) THEN PRINT "ILLEGAL":
     GOTO 170
0190 **
0200 GOSUB 440:***** paramater set
0210 FOR SECTOR=SCTST TO SCTEND
0220   PRINT :
       PRINT "TRACK=";TRACK;"  HED=";HED;" SECTOR=";SECTOR
0230   IF IO THEN PRINT #1;:
       PRINT #1;"----- TRACK=";TRACK;" HED=";HED;" SECTOR=";SECTOR
0240 GOSUB 620:***** disk read
0250 IF IOS$="8" THEN 730
0260 DADR=ADR(FDATA$(0))
0270 IF (TRACK=0)*(HED=0) THEN BYTES=127 ELSE BYTES=255
0280 FOR I=0 TO BYTES STEP 16
0290 PRINT HEX$(I);":";TAB(5);:
     IF IO=1 THEN PRINT #1;HEX$(I);":";TAB(5);
0300 DD$=""
0310 FOR II=0 TO 15
0320 DD=PEEK(DADR+I+II)
0330 PRINT HEX$(DD);" ";:
     IF IO THEN PRINT #1;HEX$(DD);" ";
0340 IF DD<$20 THEN DO$="." ELSE DO$=CHR$(DD)
0350 DD$=DD$+DO$
0360 NEXT II
0370 PRINT " ";DD$:
     IF IO=1 THEN PRINT #1;" ";DD$
0380 NEXT I
0390 NEXT SECTOR
0400 IF IO=1 THEN CLOSE #1
0410 PRINT :
     INPUT "CONTINUE(Y/N)";A$
0420 IF (A$="Y")+(A$="y")+(A$=CHR$(7)) THEN 70
0430 END
0440*
0450* I/O ROUTINE PARAMATER SET
0460*
0470 MFDD=$0533:* DRIVE ROUTINE ADDRESS
0480 AREA=$FEAB:* PARAMATER TOP ADDRESS
0490 IOSTS=$FE3C:* I/O STATUS ADDRESS
0500*
0510 POKE AREA+2,DADR-INT(DADR/256)*256:* SET BUFFER ADDRESS
0520 POKE AREA+1,$00:* DRIVE NUMBER SET
0530 IF FDDNO<>1 THEN POKE AREA+1,$01:* for drive 2
0540 DADR=ADR(FDATA$(0))
0550 POKE AREA+2,DADR-INT(DADR/256)*256:* buffer address set
0560 POKE AREA+3,INT(DADR/256)
0570 BSZ=256-1
0580 POKE AREA+4,BSZ-INT(BSZ/256)*256:* data length set
0590 POKE AREA+5,INT(BSZ/256)
0600 POKE AREA+9,$0A:* I/O errer retry count
0610 RETURN
```

```
0620%
0630% read disk
0640%
0650 POKE AREA+6,INT(TRACK):% track
0660 POKE AREA+7,INT(SECTOR):% sector
0670 POKE AREA+8,INT(HED):%      surface set
0680 CALL MFDD
0690 IO$=" "
0700 POKE ADR(IOS$)+1,PEEK(IOSTS)
0710 RETURN
0720%
0730%
0740% I/O error
0750%
0760 PRINT
0770 PRINT "%%% I/O error %%%"
0780 GOTO 400
0790%%%%%%%%  FLOPPY DISK DRIVE ROUTINE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0800%
0810%  I/O ROUTINE ADDRESS .......... $0533
0820%      Interface address
0830%              command ---------------- 0FEABH ( 1 byte  )
0840%              drive no. -------------- 0FEADH ( 1 byte  )
0850%              I/O buffer address ----- 0FEADH ( 1 byte  )
0860%              I/O data length -------- 0FEAFH ( 2 bytes )
0870%              track ------------------ 0FEB1H ( 1 byte  )
0880%              setctor ---------------- 0FEB2H ( 1 byte  )
0890%              head ------------------- 0FEB3H ( 1 byte  )
0900%              I/O retry counter ------ 0FEB4H ( 1 byte  )
0910%              I/O status ------------- 0FE3CH ( 1 byte  )
0920%
0930%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

このプログラムは，内部ルーチンをコールしているので実行させる前にテープかディスクにセーブして下さい．また，タイプミスには注意して下さい．最悪の場合には，ディスクの内容が破壊されてしまうこともあります．

このプログラムを使えばディスク上の内容をダンプすることができます．例えばドライブ1，ヘッド0，トラック1，セクタ1の内容を見たいときには次のように操作します．

```
     %%%%% Flloppy disk dump program %%%%%

 Do you use printer? (y/n)n


DRIVE :1
TRACK :1
HED   :0
SECTOR NUMBER FROM ,TO :1,1
```

RUNするとプリンターを使うかどうかを聞きますので使う場合はYを，使わない場合はNを入力して下さい．ドライブナンバー，トラック，ヘッドに対しては，それぞれダンプしたいドライブ等の数を入力して下さい．セクタは開始セクタ，終了セクタを入力して下さい．CRTには次のように出力されます．

```
          DRIVE  :1
          TRACK  :1
          HED    :0
          SECTOR NUMBER FROM ,TO :1,1

TRACK= 1  HED= 0 SECTOR= 1
00:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
10:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
20:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
30:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
40:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
50:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
60:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
70:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
80:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
90:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
A0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
B0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
C0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
D0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
E0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
F0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

CONTINUE(Y/N)
```

7-2-3　シーケンシャル・ファイル(SF)

シーケンシャル・ファイル(順編成ファイル)は，CATALOGをとったときにSVで表されます．SAVEコマンドで作成されたファイルもシーケンシャル・ファイルになります．

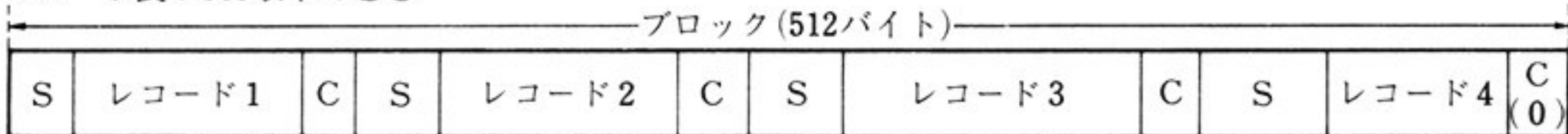
シーケンシャル・ファイルのレコード形式は次のようになっています．

- レコードは可変長で3バイトから9999バイトのレコードを取り扱う．
- ブロック長は512バイト(2セクタ)である．
- 各レコードの先頭には，レコード長が2バイトで書き込まれ，また各レコード間の区切りとして1バイトの継続マークが書き込まれる．
- レコード長が509バイト以上の時は，複数のブロックで1つのレコードを形成する．

シーケンシャル・ファイルでは，レコード長が509以下のときとそれ以外のときとでは，ブロックの構成がやや異なっています．ブロックの構成は次のようになっています．

レコード長が509以下のとき

ブロック(512バイト)

S	レコード1	C	S	レコード2	C	S	レコード3	C	S	レコード4	C(0)

レコード長が510バイト以上のとき

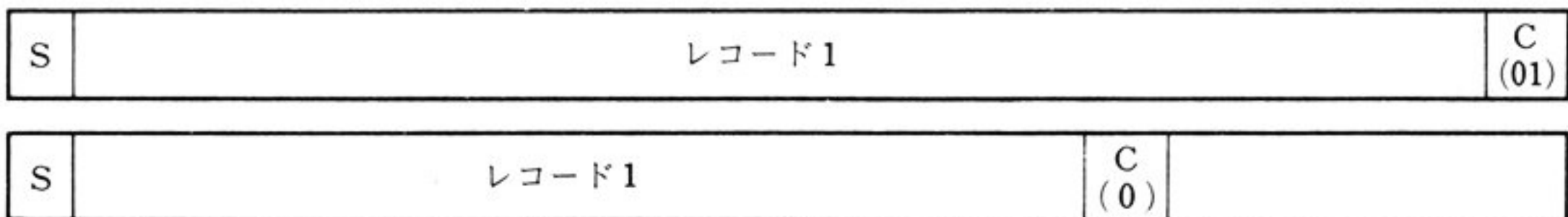



図7―2―6　シーケンシャル・ファイルのブロック構成

表中のSは2バイトのバイナリでレコード長を示していて，自分自身のバイトも含まれています．

Cは継続マークで，1バイトのバイナリです．Cはレコードが次のブロックにまたがるときは01Hになり，その他の場合は常に00Hになっています．

ファイルの取り扱い方法は次のようになります．

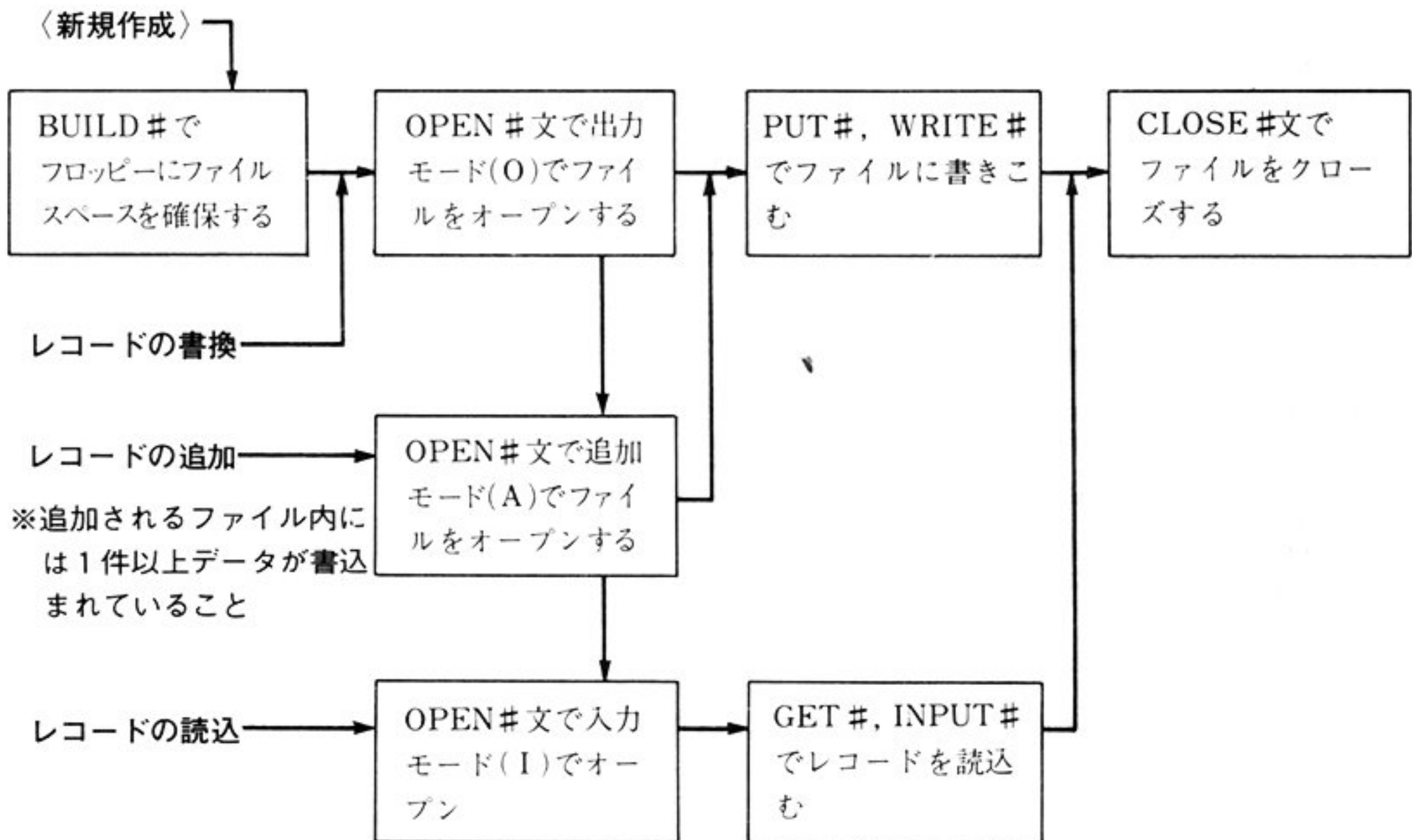



図7－2－7 シーケーシャルファイルチャート

次のプログラムでは，この方法に従ってシーケンシャル・ファイルを作成しています．

```
0010***
0020*** SV FILE DATA WRITE/READ SAMPLE
0030***
0040 WIDTH 80
0050 COLOR 7,0
0060 ERASE
0070 PRINT TAB(10,5);"*** SV FILE W/R SAMPLE ***"
0080 PRINT TAB(15);"SET DISK TO FD2"
0090*
0100* BUILD AND OPEN FILE
0110*
0120 BUILD #"FD2","TESTSV","SV",210,,,5,150
0130 OPEN #"FD2","TESTSV",1,"O"
0140 GOTO 190
0150'----- PURGE OLD FILE -----
0160 PURGE #"FD2","TESTSV"
0170 GOTO 120
0180*
0190* MAKE DATA
0200*
0210 DIM D1$*200(0),D2$*10(0)
0220 D1$(0)=""
0230 FOR I=1 TO 200
0240   D1$(0)=D1$(0)+CHR$(I)
0250 NEXT I
0260 D2$(0)="ABCDEFGHIJ"
0270*
0280* PRINT DATA TO FILE
0290*
0300 PRINT :
```

```
     PRINT TAB(10);"NOW WRITE DISK"
0310 I=1
0320 S$=" ; "
0330 PUT #1,I,S$,D1$(0)
0340 I=I+1
0350 PUT #1,I,S$,D2$(0)
0360 I=I+1
0370 PUT #1,I,S$,D1$(0)
0380 CLOSE #1
0390*
0400* READ DATA FROM FILE
0410*
0420 PRINT :
     PRINT TAB(10);"---NOW READING"
0430 OPEN #"FD2","TESTSV",1,"I"
0440 GET #1,I,S$,D1$(0)
0450 COLOR 4
0460 PRINT I,S$,D1$(0)
0470 GET #1,I,S$,D2$(0)
0480 COLOR 5
0490 PRINT I,S$,D2$(0)
0500 GET #1,I,S$,D1$(0)
0510 COLOR 6
0520 PRINT I,S$,D1$(0)
0530 CLOSE #1
0540 COLOR 7
0550 END
```

このファイルを前述のディスクダンプ・プログラムによってダンプすると，次のようにディスクに書き込まれていることがわかります．なお，四角で囲っているものは前述のレコード長と継続ポインタです．

```
       レコードデータ長
       |
-----  TRACK= 30 HED= 1 SECTOR= 15
00:  [D3 00] 40 01 00 00 00 03 20 3A 20 C8 01 02 03 04    t.@..... : ネ....
10:   05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14    ................
20:   15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24    ........... !"#$
30:   25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34    %&'()*+,-./01234
40:   35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44    56789:;<=>?@ABCD
50:   45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54    EFGHIJKLMNOPQRST
60:   55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64    UVWXYZ[¥]^_`abcd
70:   65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74    efghijklmnopqrst
80:   75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84    uvwxyz{¦}~▁▂▃▄▅
90:   85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94    ▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔
A0:   95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4    ─│▕┌┐└┘╭╮╰╯ 。「」、
B0:   A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4    ・ヲァィゥェォャュョッーアイウエ
C0:   B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4    オカキクケコサシスセソタチツテト
D0:   C5 C6 C7 C8 00 [15 00] 40 02 00 00 00 03 20 3A 20    ナニヌネ...@..... :
E0:   0A 41 42 43 44 45 46 47 48 49 4A 00 [D3 00] 40 03    .ABCDEFGHIJ.t.@.
F0:   00 00 00 03 20 3A 20 C8 01 02 03 04 05 06 07 08    .... : ネ........

-----  TRACK= 30 HED= 1 SECTOR= 16
00:   09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18    ................
10:   19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28    ....... !"#$%&'(
20:   29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38    )*+,-./012345678
30:   39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48    9:;<=>?@ABCDEFGH
40:   49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58    IJKLMNOPQRSTUVWX
50:   59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68    YZ[¥]^_`abcdefgh
60:   69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78    ijklmnopqrstuvwx
70:   79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88    yz{¦}~▁▂▃▄▅▆▇█
80:   89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98    ▏▎▍▌▋▊▉┼┴┬┤├▔─│▕
90:   99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8    ┐└┘╭╮╰╯ 。「」、・ヲァィ
A0:   A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8    ゥェォャュョッーアイウエオカキク
B0:   B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8    ケコサシスセソタチツテトナニヌネ
C0:   00 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
D0:   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
E0:   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
F0:   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

7-2-4　ランダムアクセス・ファイル(RF)

ランダムアクセス・ファイル(直編成ファイル)は，固定長コードのレコードによって作られたファイルで，SAVESによるファイルもランダムアクセス・ファイルです．

ランダムアクセス・ファイルは，次のようなレコード形式で記録されています．

・固定長レコード長は3～256バイトである．

・ブロックサイズは512バイト(2セクタ)である．

・コントロール情報は付加されない．

全て同一レコード長で作成される．空領域はレコードの長さより短い

図7—2—8　ランダムファイルのレコード形式

ランダムアクセス・ファイルを取り扱う手順は次のようになります．

〈新規作成〉　〈レコードの読取/書込〉　〈レコードの追加〉

BUILD＃

OPEN出力モード

入力/出力/入出力モードでOPEN

追加モードでOPEN

呼出か？

NO

Yes

呼び出すレコードのポジション設定 SEARCH＃

レコードの入出力処理

ファイルをCLOSE

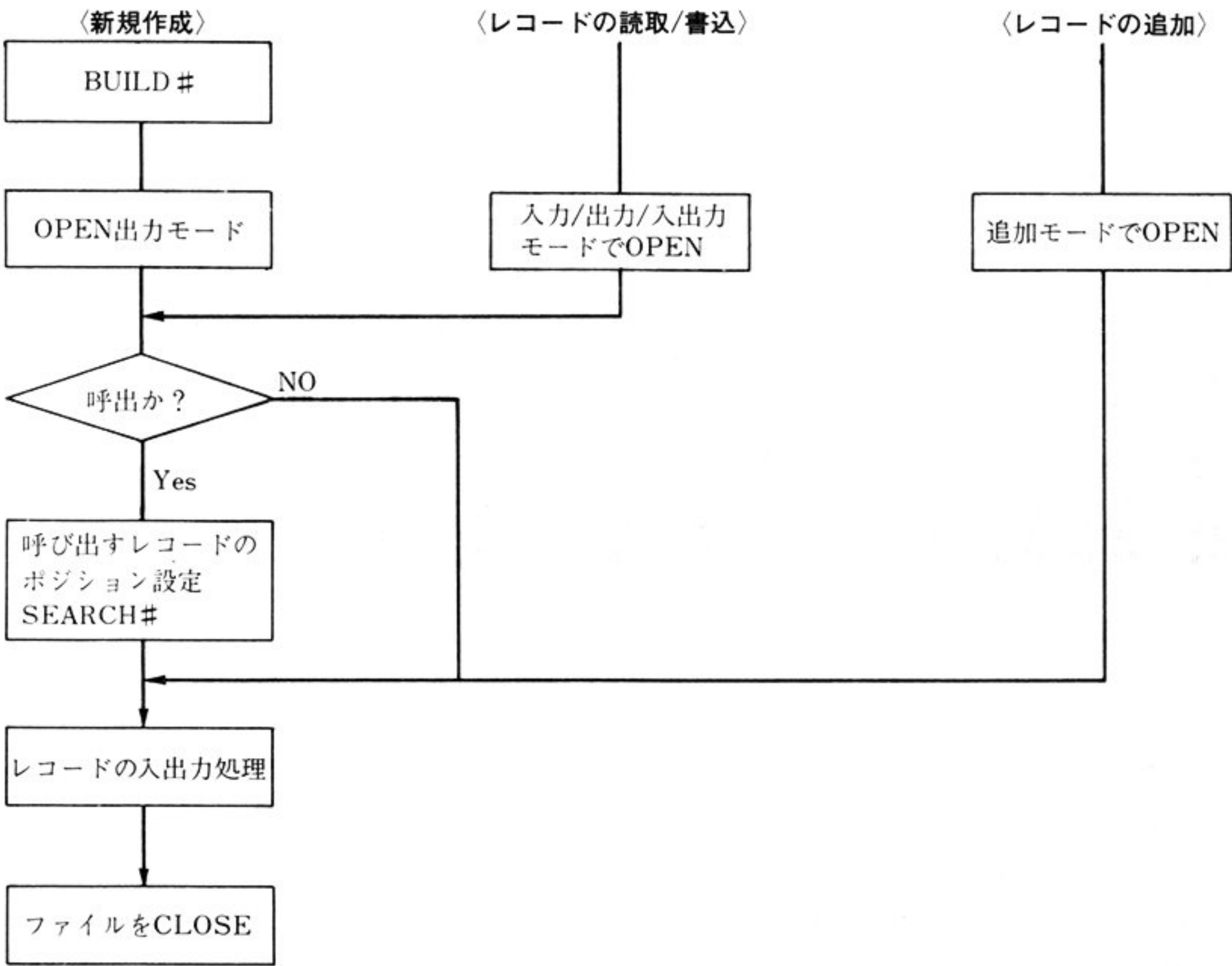



図7—2—9　ランダムファイルチャート

次のプログラムはランダムアクセス・ファイルのサンプルです.

```
0010***
0020***    RF FILE READ/WRITE SAMPLE
0030***
0040 BUILD #"FD2";"RFFILE","RF",60,,,5,60:* BUild file space for FD2
0050 GOTO 80
0060 PURGE #"fd2","RFFILE"
0070 GOTO 40
0080 OPEN #"FD2","RFFILE",1,"I/O":*          open file as input/output
0090 DIM FDATA$*50(0):*                      string space 50
0100 FDATA$(0)=""
0110 FOR I=40 TO 99
0120   FDATA$(0)=FDATA$(0)+CHR$(I)
0130 NEXT I
0140 FOR I=1 TO 10
0150   SEARCH #1,"SR",I:*                    positioning for write
0160   PUT #1,I,FDATA$(0):*                  write data
0170 NEXT I
0180 SEARCH #1,"SR",2:*                      positioning for read
0190 GET #1,I,FDATA$(0):*                   read data
0200 PRINT I,FDATA$(0)
0210 GET #1,I,FDATA$(0):*                    read data
0220 PRINT I,FDATA$(0)
0230 CLOSE #1
0240 END
```

ランダムアクセス・ファイルは，フロッピィには次のように書き込まれています.

```
                                          レコード(60バイト)
----- TRACK= 30 HED= 1 SECTOR= 14
00:  40 01 00 00 00 32 28 29 2A 2B 2C 2D 2E 2F 30 31   @....2()*+,-./01
10:  32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41   23456789:;<=>?@A
20:  42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51   BCDEFGHIJKLMNOPQ
30:  52 53 54 55 56 57 58 59 20 20 20 20 40 02 00 00   RSTUVWXY    @...
40:  00 32 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35   .2()*+,-./012345
50:  36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45   6789:;<=>?@ABCDE
60:  46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55   FGHIJKLMNOPQRSTU
70:  56 57 58 59 20 20 20 20 40 03 00 00 00 32 28 29   VWXY    @....2()
80:  2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39   *+,-./0123456789
90:  3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49   :;<=>?@ABCDEFGHI
A0:  4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59   JKLMNGPQRSTUVWXY
B0:  20 20 20 20 40 04 00 00 00 32 28 29 2A 2B 2C 2D       @....2()*+,-
C0:  2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D   ./0123456789:;<=
D0:  3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D   >?@ABCDEFGHIJKLM
E0:  4E 4F 50 51 52 53 54 55 56 57 58 59 20 20 20 20   NOPQRSTUVWXY
F0:  40 05 00 00 00 32 28 29 2A 2B 2C 2D 2E 2F 30 31   @....2()*+,-./01
```

7-2-5 インデクス・シーケンシャル・ファイル(ISF)

インデクス・シーケンシャル・ファイル(索引順編成ファイル)は，レコードが書き込まれた順番に関係なくレコードが持つキー(索引)で管理できるようになっているファイルで，他のBASICには見られないファイルです.

索引ブロックは，レコードが収容されるデータプログラムのポインタと，そのデータブロックをたどるためのキーから構成されています.

概念図を次に示します.

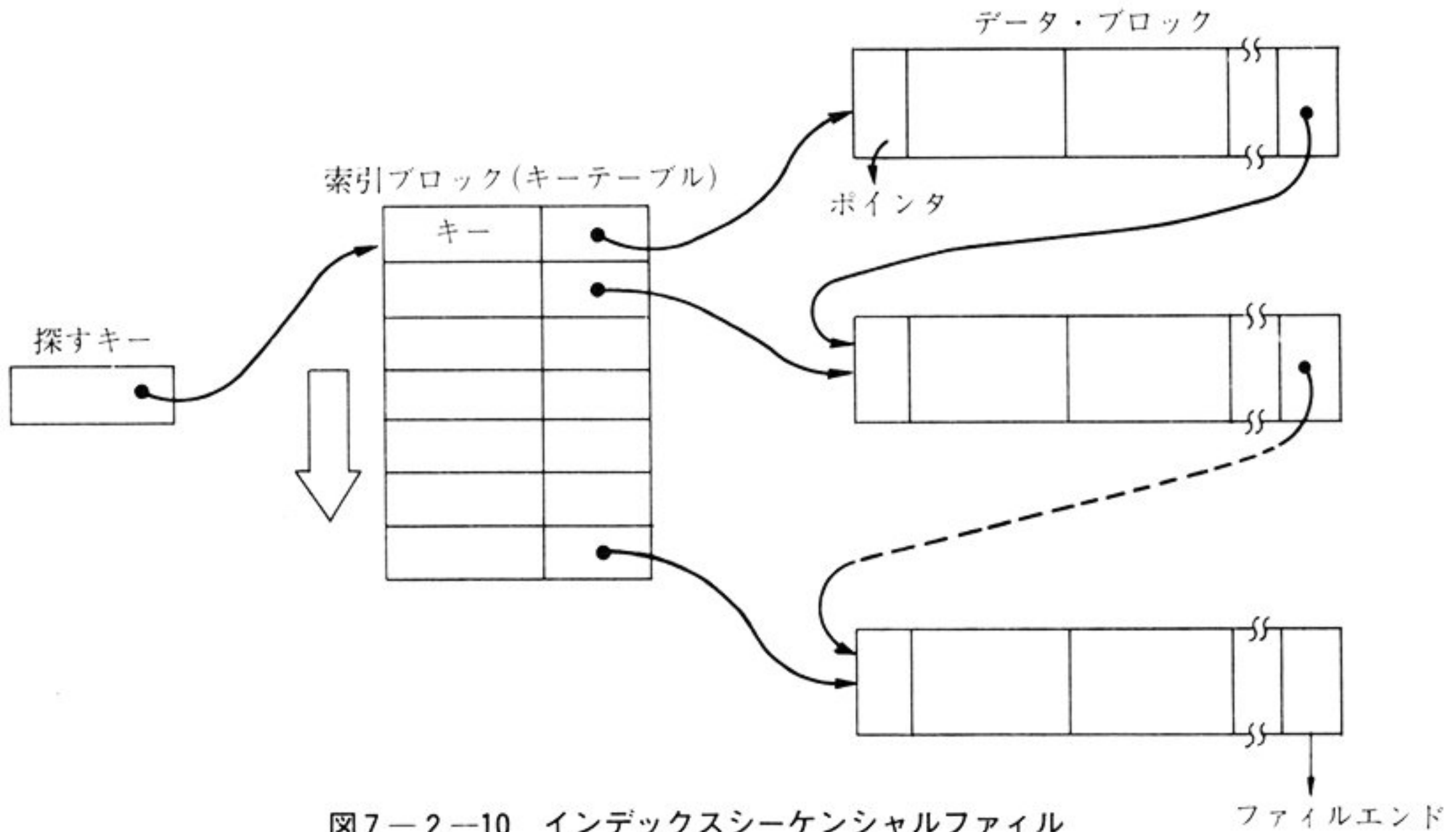



図7—2—10 インデックスシーケンシャルファイル

ISFはレコードをキーの値によって呼び出すので,ファイルの効率が良いという特徴があります.したがってISFはレコードが頻繁に挿入追加,削除,更新されるようなファイルに使うと効果的です.

索引ブロックの大きさは512バイトで,各データブロック中の最終キーとそのデータブロックへのポインタをもっています.索引ブロックは主索引ブロックと副索引1ブロック,副索引2ブロックから構成されています.それぞれのブロックをPIX,SIX1,SIX2と呼びます.

図7—2—11 ブロック構成

SIXのレベルが1まで2までかはデータ件数によって自動的に決定されます.PIXはSIXへのブロックポインタを持ち,SIXはデータブロックへのブロックポインタを持っています.ISFの構造を図にすると次のようになります.

データブロック図

探すキー

主索引ブロック

副索引ブロック

(最大７ブロック)

副索引(２)ブロック

(最大31×７ブロック)

データブロック

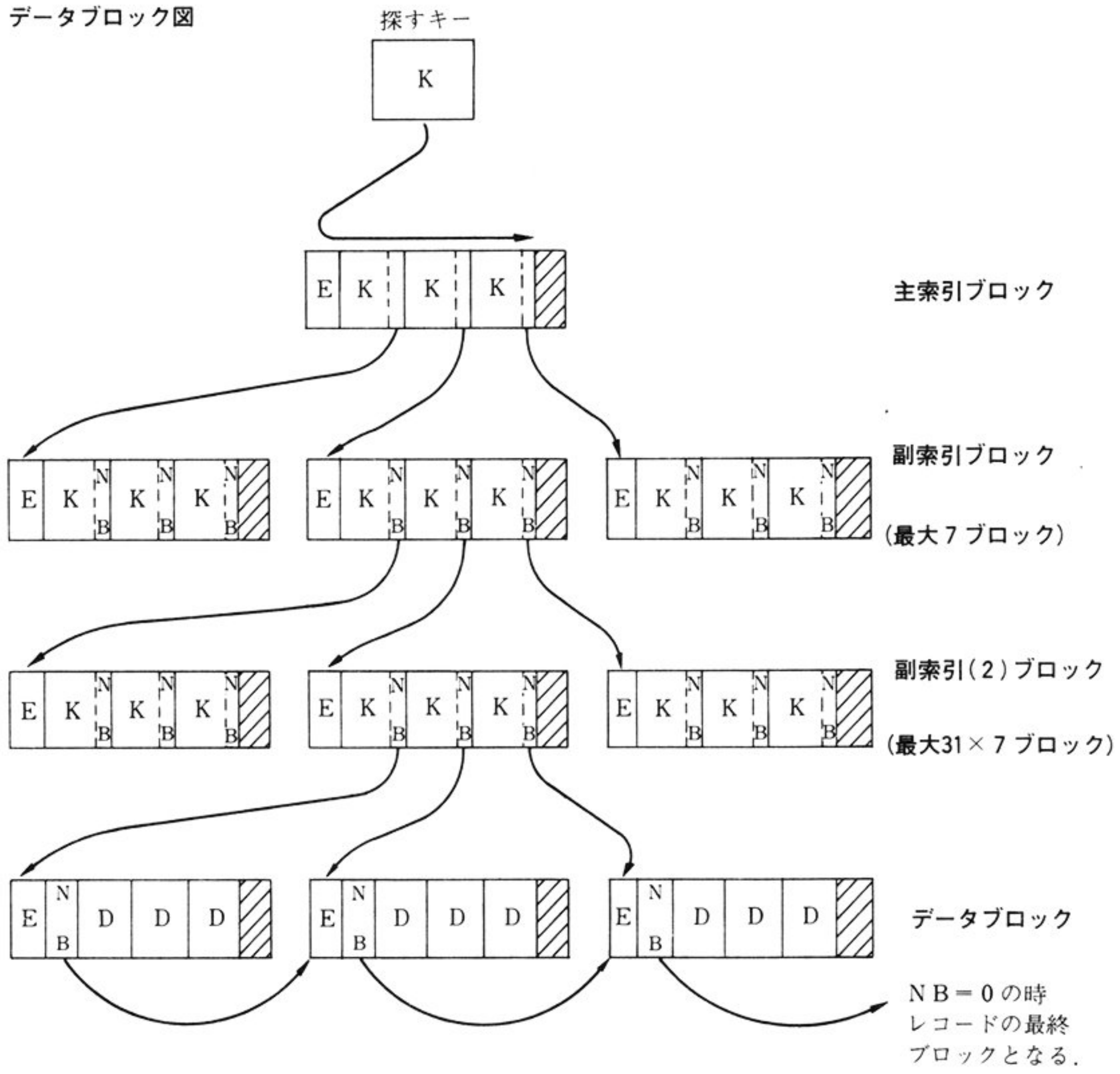
ＮＢ＝０の時
レコードの最終
ブロックとなる.

記号	タイプ	バイト数	内容
E	バイナリ	1	1ブロック内のデータまたはキーのエントリ数
NB	バイナリ	1	次のキーまたはデータブロックのブロックポインタ値
K		14	キーの内容
D		128	データの内容

図７—２—12　インデックス・シーケンシャルファの構造

索引ブロックは次のような構成になっています.

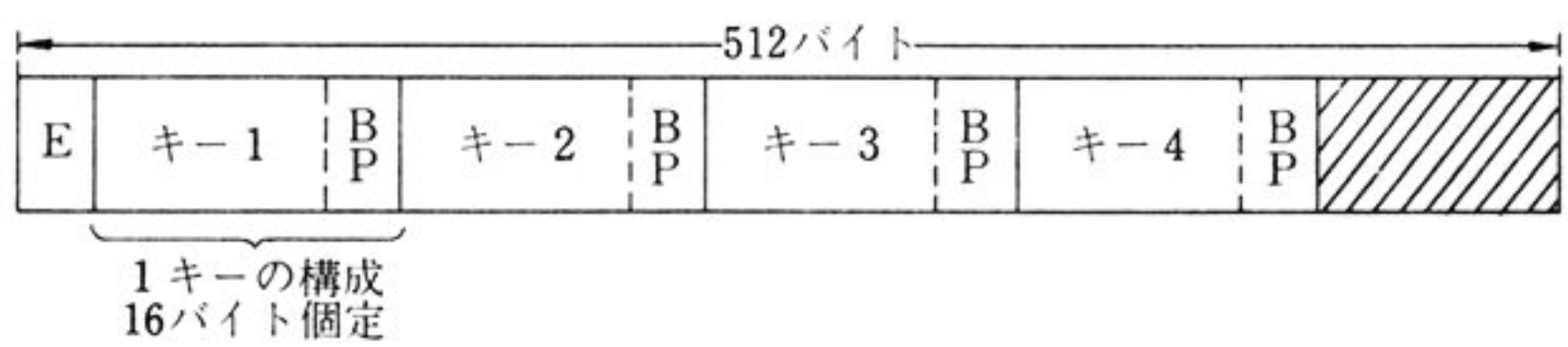



図７—２—13　索引ブロック

この図からもわかるように，1ブロックは512バイト（2セクタ）に固定されています．Eは1ブロック内のキーの数を示す1バイトの16進数であり，キーはブロックポインタ(BP)で示されるブロック中の最終のキー値で，これは14バイトに固定されています．最後のキーは14バイトすべてFFHとなります．またBPはBOE(ファイル開始位置)を1とした相対的な位置で示しています．

レコードは512バイトのブロックに割り当てを施されて収容されます．レコードは3～254バイトの固定長です．

図中でEは，そのブロックのレコード数（1バイトバイナリ）でBPは次のデータのブロックポインタで，ブロック内のデータはキーの昇順に並んでいます．

ISFの取り扱いは図7-2-15のようになります．

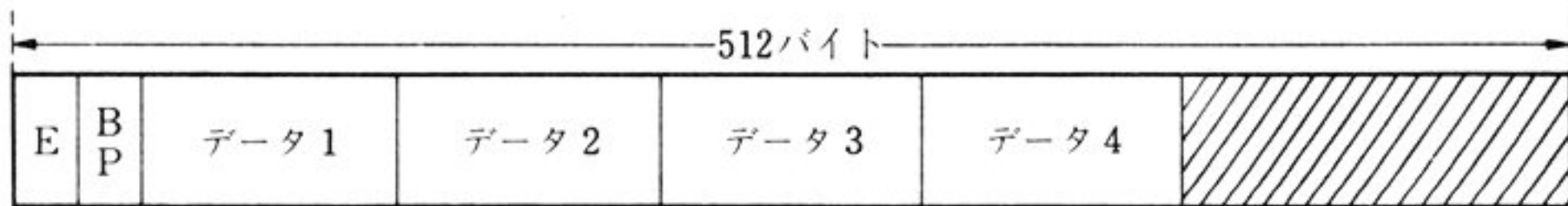



図7－2－14　データ・ブロック

次のプログラムはISFの作成のサンプルです．

```
0010 ****
0020 ****  IS FILE DATA WRITE/READ
0030 ****
0040 WIDTH 80:
     ERASE
0050 PRINT TAB(10,5);"*** IS FILE SAMPLE ***"
0060 PRINT TAB(15,6);"Set floppy disk for FD2"
0070*  build and open          *
0080 BUILD #"FD2","testis","IS",120,2,14,30,100
0090 GOTO 120
0100 PURGE #"fd2","testis":*  kill same name file.
0110 GOTO 80
0120 DIM KEY$*14(0),DATA1$*104(0)
0130 OPEN #"FD2","testis",1,"I/O"
0140 DATA1$(0)=""
0150*  make data and print data  *
0160 PRINT SPC(10);" Write data to fd2"
0170 FOR I=65 TO 168
0180    DATA1$(0)=DATA1$(0)+CHR$(I)
0190 NEXT I
0200 FOR I=65 TO 74
0210    KEY$(0)=CHR$(I)+"000000000000"+CHR$(I)
0220    SEARCH #1,"AK",KEY$(0)
0230    PUT #1,KEY$(0),DATA1$(0)
0240 NEXT I
0250*  read data               *
0260 PRINT SPC(10);" Read data from disk":
     PRINT
0270 FOR I=70 TO 74
0280 COLOR I-68
0290    KEY$(0)=CHR$(I)+"000000000000"+CHR$(I)
0300    SEARCH #1,"SK",KEY$(0)
0310    GET #1,KEY$(0),DATA1$(0)
0320    PRINT KEY$(0),DATA1$(0)
0330 NEXT I
0340 CLOSE #1
0350 END
```

SEARCH＃文のファイクション

値	意味
AK	レコード追加
DK	レコード消去
SK	レコード読み込み
＃R	レコードの変更

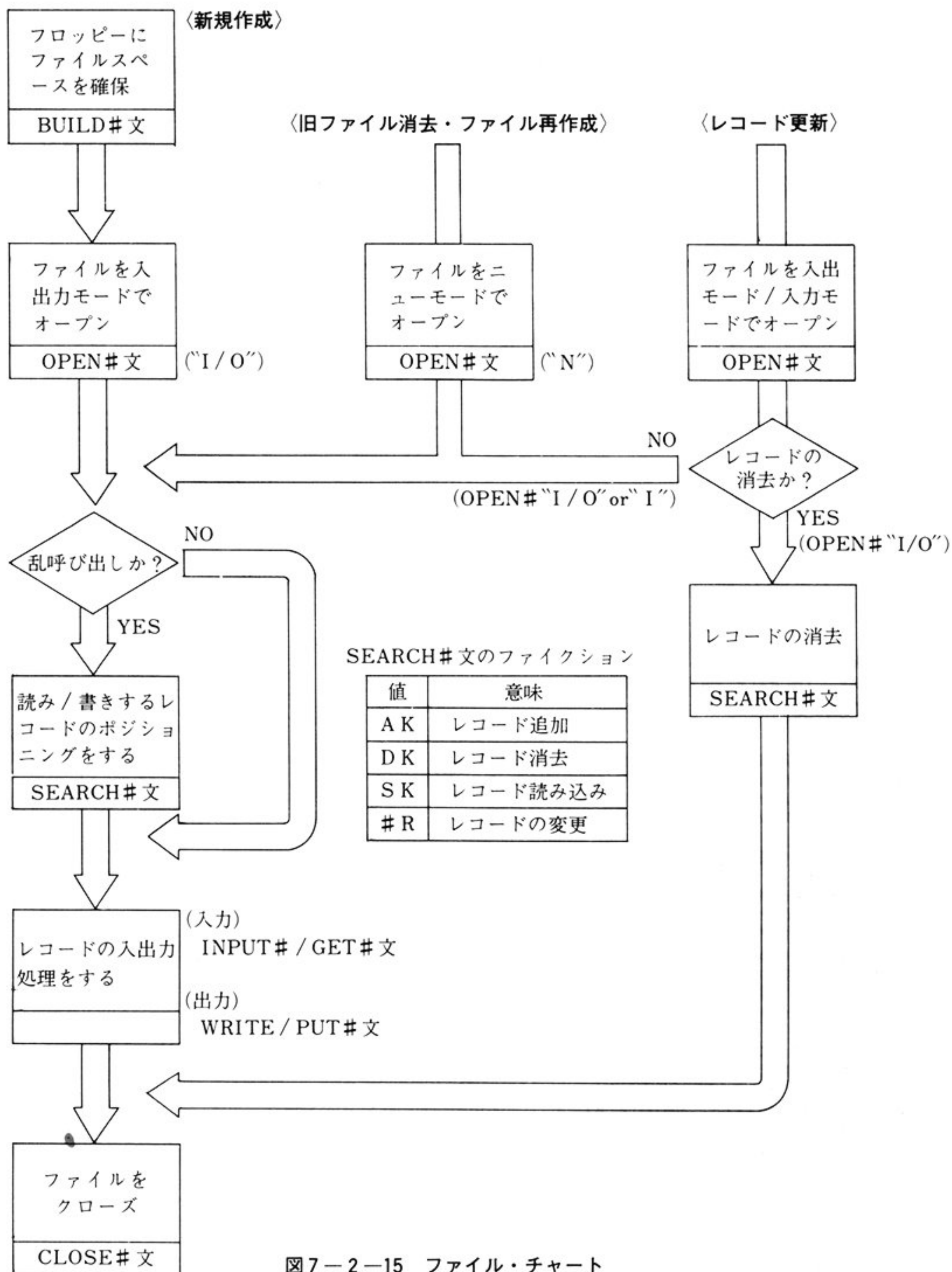



図7―2―15　ファイル・チャート

7-3　グラフィック

OA-BASICではテキストモード，セミグラフィックモード，ファイングラフィックモードの3

種のスクリーンモードが定義されています．スクリーンモードの切換はSCREEN命令で行います．SCREENに関しての詳しいことは，T-BASICとほとんど同じなので第3章を参照して下さい．

ここでは，OA-BASICのグラフィック機能について説明します．

7-3-1　T-BASICとの比較

OA-BASICとT-BASICのグラフィック命令は，パラメータやオペレーションがやや異なっています．漢字の出力等も含めたグラフィック命令の比較表を次に示します．

この中で注意すべき点は，SCREEN 2のときのLINEとPSETです．T-BASICではLINE命令とPSETのカラーコードは0のときには，背景色でその他のときには書き込まれた行の色で描かれますが，OA-BASICではカラーアトリビュートデータを書き込むスペースがあれば色コードは有効になります．アトリビュートを書き込むスペースが無い場合は左方向の表示色で描かれます．これは，PRINTでも同様で，キャラクタに着色したいときには，アトリビュート・キャラクタを書き込むスペースがあることが条件になります．

また，01H～1BHの特殊キャラクタは，T-BASICではコントロールキャラクタ(例えばCHR$(12)は画面クリア)ですが，OA-BASICでは特殊な機能を示すことはありません．

しかし01H～1FHのキャラクタは，プリンタでハードコピーをとったときには出力されず，かわりにスペースが出力されます．

図7－3－1　OA-BASICとT-BASICの比較

	意　味	T-BASIC	OA-BASIC	備　考
スクリーンモード	表示文字数を変える．	WIDTH　x (0<x≦255)	WIDTH x, y (x=36, 80 / y=20, 25)	T-BASICでは行数を変えることはできない．
	スクリーンのグラフィックモードを変える．	SCREEN　n (n=0, 1, 2)	Tと同じ	
キャラクタ表示など	画面のクリア	CLS	ERASE	
	カーソルを(x, y)に移動．	LOCATE　x, y	PRINT　TAB(x, y)；	
	カーソルの表示位置のx座標，y座標を知る．	x=POS(0) y=CSRLIN	――	
	(x, y)のキャラクタコードを知る．	n=SCREEN(x, y)	――	
	キャラクタの反転表示．	――	SFLG 8, 1	テキストモードのみ．SFLG 8, 0で通常の表示．
	キャラクタで線を引く．	――	LINE(x_0, y_0) −(x_1, y_1), A$	文字変数や" で囲まれたキャラクタで描線する．

グラフィック	点を打つ.	PSET(x, y), 表示色	PSET(x, y, 表示色)	T-BASICではSCREEN2のとき第3パラメータは無視される.
	描線	LINE(x_0, y_0) − (x_1, y_1) [, 表示色[, {B / BF}]]	LINE(x_0, y_0) − (x, y), {PSET / PRESET} [, 表示色[, {"B" / "BF"}]]	上と同じ
	画面情報を配列に格納. 配列を画面に出力する.	GET@(x_0, y_0) −(x_1, y_1), A% PUT@(x_0, y_0), A%		A%は配列名. (T-DISK-BASIC.) SCREEN2でも同じ.
漢字出力	漢字の出力 (SCREEN␣2)	PUT@(x, y), KANJI(&K1601)	PRINT &1601	この例では"亜"を表示する.

7-3-2 アトリビュート

OA-BASICのLINE文では, カラーアトリビュートデータを書き込みます. CRT上の同じ行で複数の色を使いたいときには, 色を変化させたいところで15ドット以上の間隔をとり, アトリビュートデータを書き込めるようにする必要があります. 次のプログラムでは, そのような条件を考慮したグラフィックのサンプルです.

```
0010 WIDTH 80:
     SCREEN 2:
     ERASE
0020 COLOR 7,1
0030 C=1
0040 FOR I=199 TO 0 STEP -8
0050 C=C+1:
     IF C>7 THEN C=1
0060 LINE (0,I)-(639,I),PSET,2
0070 NEXT I
0080 FOR J=1 TO 7
0090 FOR I=199-J TO 0 STEP -8
0100 LINE (I*2,I)-(I*2+20,I),PSET,C
0110 C=C+1:
     IF C>7 THEN C=1
0120 LINE (639-I*2,I)-(639-I*2-20,I),PSET,C
0130 NEXT I
0140 NEXT J
0150 FOR J=1 TO 7
0160 FOR I=199-J TO 0 STEP -8
0170 IF I*2-16<20 THEN 210
0180 X1=639-I*2-20:
     X2=I*2
0190 IF X1<X2 THEN X=X1-16 ELSE X=X2-16
0200 LINE (X-20,I)-(X,I),PSET,6
0210 X1=639-I*2:
     X2=I*2+20
0220 IF X1>X2 THEN X=X1+16 ELSE X=X2+16
0230 IF X>639 THEN 250
0240 LINE (X,I)-(X+20,I),PSET,5
0250 NEXT I
0260 NEXT J
0270 J=2
0280 FOR I=100 TO 36 STEP -1
0290   J=J+1
       IF J>7 THEN J=0
```

```
0300    X1=(100-I)^(5/2)-I^2
0310    X1=X1/100
0320    LINE (320-X1,I)-(320+X1,I),PSET,J
0330 NEXT I
0340 END
```

7-4 漢字入出力

OA-BASICで漢字を使うときには，フロッピィディスクの@KJPATというランダムアクセスファイルから漢字パターンのデータを読み出して使用します．漢字パターンは1字につき16×16ドットのデータとなっています．1字分のデータの形式はT-BASICと同じなので第6章を参照して下さい．

7-4-1 漢字パターン・ファイル

漢字パターンファイル@KJPATは，ディスクの上の492セクタを占有し1836字分のデータを持っています．このファイル上に次のように漢字パターンを格納しています．また，利用者が漢字パターンを更新することによって外字の登録も84区から94区までは可能になっています．

漢字パターンファイルは次の図のような順番で格納されています．

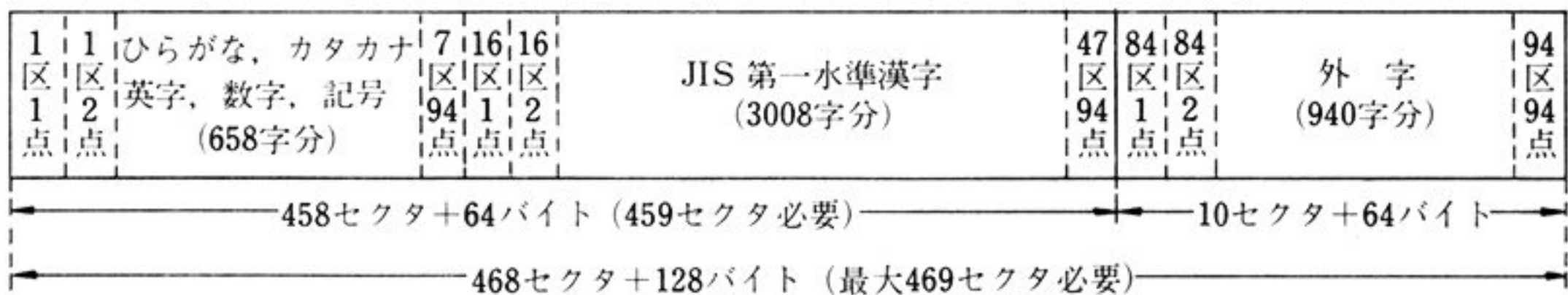



図7－4－1 漢字パターンファイルの概略

漢字パターンファイルは，1文字文のパターンを1レコードとしたシーケンシャルファイルです．パターン格納の相対位置をファイルの開始位置を1とすると次のようになります．

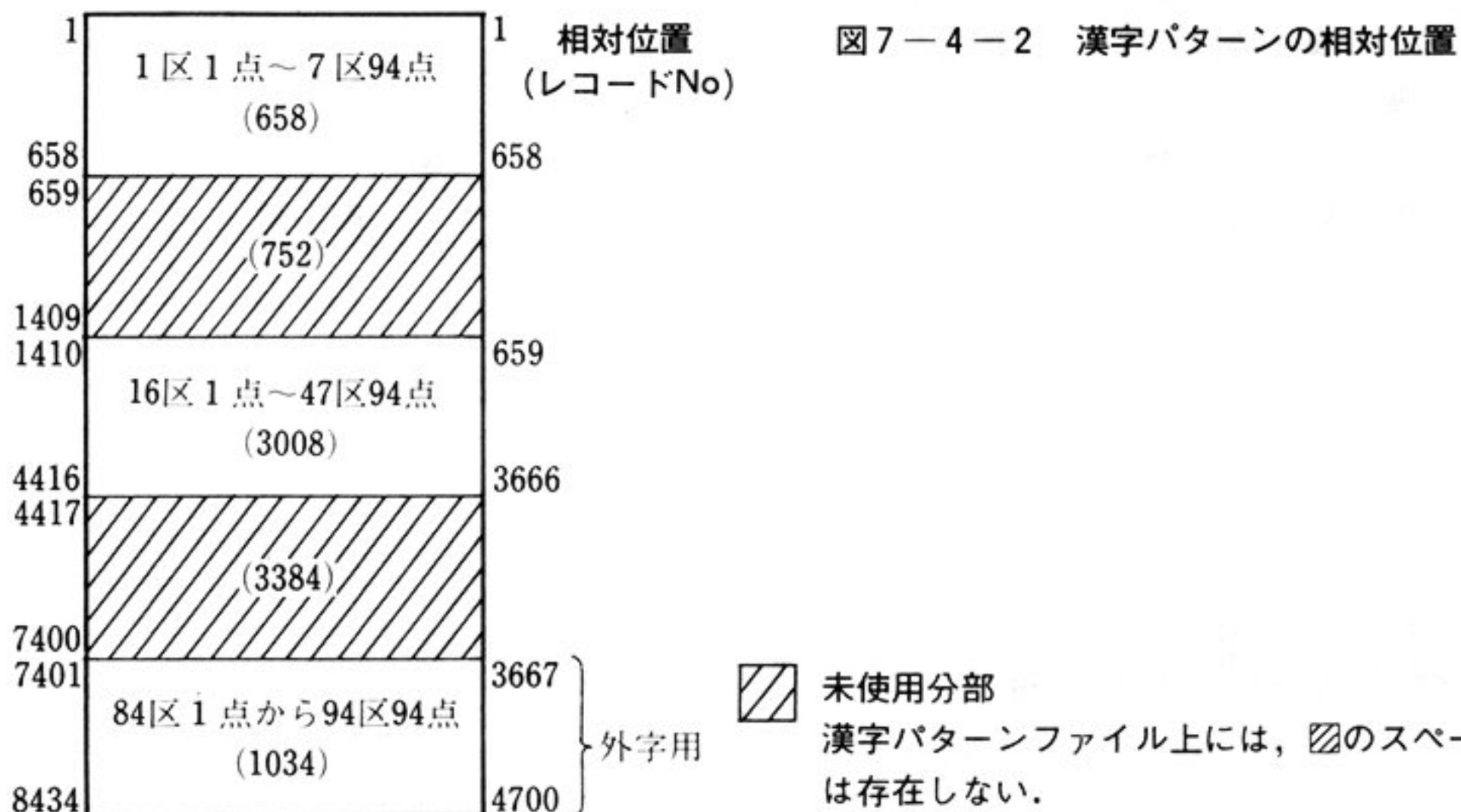



図7－4－2 漢字パターンの相対位置

〈区点番号による相対位置の算出〉

相対位置＝(区番号－1)＊94＋点番号－X

〈区点，コードによる相対位置の算出〉

相対位置＝(区コード－33)＊94＋(点コード－32)－X

区番号(区コード)	Xの位
1から7 (33から39)	0
16から47 (48から94)	252
84から94 (116～126)	4136(752＋1034)

7-4-2　漢字パターン入出力

漢字パターンの入出力には文字列長が付加されるためGET#，PUT#文を使うことはできません．必ずINPUT#/WRITE#で入出力を行う必要があります．WRITEは，USINGを付けて語長を32バイトにする必要があります．

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	データ値(10進) 1～8	データ値(10進) 9～16
1													●	●			0	12
2												●		●			0	20
3												●		●			0	20
4											●			●			0	36
5											●			●			0	36
6											●			●			0	36
7										●			●				0	72
8									●				●				0	136
9									●				●				0	136
10								●					●				1	8
11					●	●	●	●	●	●	●	●	●	●	●		15	254
12			●	●			●						●				50	8
13		●				●							●				68	8
14	●				●									●			136	4
15		●	●	●											●	●	112	3
16																	0	0

データは32バイト

図7－4－3　外字パターンの作成例

次のプログラムは，漢字パターンの入力の例です．

```
*LIST
0010 PRINT &0348033303510347;
0020 PRINT &0348034103330446;
0030 PRINT &2033279029484647
0040 END
*RUN
PASOPIAの漢字出力
```

次のプログラムは，外字の登録のサンプルです．漢字パターンを登録する位置は区番号もしくは区点コードで入力する必要があります．

```
1000***
1010*** KANJI FONT EDITOR PROGRAM
1020***
1030***         FOR OA-BASIC
1040***
1050 WIDTH 36,24
1060 SCREEN 2
1070 COLOR 7,0
1080 ERASE
1090 *START
1100 GOSUB 1740:* CRT SET
1110 GOSUB 1550:* INITIALIZE
1120 GOSUB 1210:* FONT EDIT
1130 PRINT TAB(0,21);"END OK (Y/N)";
1140 INPUT A$:
     IF (A$="N")+(A$="n") THEN GOSUB 1740:
     GOSUB 2190:
     GOTO 1120
1150 GOSUB 2290:* SAVE FD
1160 GOSUB 2740:* DYSPLAY FONT
1170 COLOR 6
1180 INPUT "CONTINUE ( Y/N ):";W$
1190 IF (W$="Y")+(W$="y")+(W$=CR$) THEN CLEAR :
     GOTO 1050
1200 END
1210*
1220* EDIT FONT
1230*
1240 GOSUB 2190:* FONT OUTPUT
1250 XPOS=1:
     YPOS=1:* SET INITIAL POINT X,Y
1260 SFLG 1,0:** COURSOL OFF
1270 PRINT TAB(5,3+YPOS);FONT$(YPOS)
1280 PRINT TAB(4+XPOS,3+YPOS);
1290 SFLG 1,1:** COURSOL ON
1300 W$=KIN$
1310 IF NULL THEN 1300
1320 SFLG 1,0:*** COURSOL OFF
1330 IF W$=CR$ THEN RETURN
1340 IF (W$=RARROW$)*(XPOS<16) THEN XPOS=XPOS+1
1350 IF (W$=LARROW$)*(XPOS>1) THEN XPOS=XPOS-1
1360 IF (W$=UARROW$)*(YPOS>1) THEN YPOS=YPOS-1
1370 IF (W$=DARROW$)*(YPOS<16) THEN YPOS=YPOS+1
1380 IF W$=" " THEN MID$ (FONT$(YPOS),XPOS,1)="●":
     PSET (XST+XPOS-1,YST+YPOS-1)
1390 IF W$="0" THEN MID$ (FONT$(YPOS),XPOS,1)="o":
     PRESET (XST+XPOS-1,YST+YPOS-1)
1400 IF W$=PF$(8) THEN GOTO 1430:*** FONT CLEAR CKECK***
1410 IF W$=PF$(1) THEN GOSUB 1930:* HELP
1420 GOTO 1260
1430*
1440* FONT CLEAR CHECK
1450*
1460 COLOR 6
1470 PRINT TAB(0,21);"FONT CLEAR OK (Y/N)";
1480 INPUT A$
1490 COLOR 7
1500 PRINT TAB(0,21);SPC(35)
1510 IF NULL THEN 1480
1520 IF (A$="Y")+(A$="y") THEN GOSUB 1550:
     GOSUB 1740:
     GOTO 1210
1530 IF (A$="N")+(A$="n") THEN GOTO 1260
1540 GOTO 1480
1550*
1560*  INITIALIAE
1570*
```

```
1580 DIM FONT$%16(16)
1590 FOR I=1 TO 16
1600   FONT$(I)=STRING$(16,"o")
1610 NEXT I
1620 FONTNO=1
1630%%%%%CONTROL CHARACTOR
1640 CR$=CHR$(8)
1650 UARROW$=CHR$($1E)
1660 DARROW$=CHR$($1F)
1670 LARROW$=CHR$($1D)
1680 RARROW$=CHR$($1C)
1690 DIM PF$%1(8)
1700 FOR I=1 TO 8
1710   PF$(I)=CHR$($10+I)
1720 NEXT I
1730 RETURN
1740%
1750% CRT INITIALIZE
1760%
1770 ERASE
1780 PRINT "<<  KANJI EDITOR for T-BASIC >>"
1790 PRINT TAB(4,3);"┌";STRING$(16,"─");"┐"
1800 FOR I=4 TO 19
1810   PRINT TAB(4,I);"|";SPC(16);"|"
1820 NEXT I
1830 PRINT TAB(4,20);"└";STRING$(16,"─");"┘"
1840 PRINT TAB(28,3);"FONT"
1850 COLOR 4
1860 XST=230
1870 YST=40
1880 COLOR 4
1890 PRINT TAB(4,23);"ｷｰ ﾉ ｾﾂﾒｲ ﾊ [PF1] ﾃﾞｽ.";
1900 COLOR 7
1910 LINE (XST-1,YST-1)-(XST+17,YST+17),PSET,7,"B"
1920 RETURN
1930%
1940% COMMAND INSTRUCTION
1950%
1960 ERASE
1970 COLOR 4
1980 PRINT ">> COMMAND IS FOLLOWING."
1990 COLOR 6
2000 PRINT "      ";UARROW$;TAB(10);": coursol up."
2010 PRINT "      ";DARROW$;TAB(10);": coursol down."
2020 PRINT "      ";RARROW$;TAB(10);": coursol right."
2030 PRINT "      ";LARROW$;TAB(10);": coursol left."
2040 PRINT
2050 COLOR 5
2060 PRINT "  [space]";TAB(10);": set dot."
2070 PRINT "  [ 0 ]";TAB(10);": reset dot."
2080 COLOR 6
2090 PRINT
2100 PRINT "  [return]";TAB(10);": end of edit mode."
2110 PRINT "  [ PF 1 ]";TAB(10);": output this message."
2120 PRINT "  [ PF 8 ]";TAB(10);": font clear."
2130 PRINT
2140 INPUT "OK >>HIT RETURN KEY";A$
2150 COLOR 7
2160 GOSUB 1740:% CRT INITIALIZE
2170 GOSUB 2190:% output font for crt
2180 RETURN
2190%
2200% FONT OUTPUT
2210%
2220 FOR I=1 TO 16
2230   PRINT TAB(5,3+I);FONT$(I)
2240   FOR J=1 TO 16
2250     IF MID$(FONT$(I),J,1)="●" THEN PSET (XST+J-1,YST+I-1)
2260   NEXT J
2270 NEXT I
2280 RETURN
2290%
2300% SAVE FD
```

```
2310*
2320 ERASE
2330 PRINT " MOUNT SYSTEM DISK FOR DRIVE #1"
2340 INPUT "      AND HIT RETURN ";A$
2350 *
2360***** MAKE KANJI DATA
2370 KNJPAT$=""
2380 FOR J=1 TO 16
2390   FOR I=1 TO 9 STEP 8
2400     W=0
2410     FOR K=1 TO 8
2420       IF MID$(FONT$(J),I+K,1)="●" THEN W=W+2^(8-K)
2430     NEXT K
2440   KNJPAT$=KNJPAT$+CHR$(W)
2450   NEXT I
2460 NEXT J
2470***** INPUT KANJI CODE
2480 PRINT "PLEASE INPUT KUTEN OR CODE"
2490 PRINT "  NO      ... 1"
2500 INPUT "  CODE    ... 2 :";A
2510 IF (A=1)+(A=2)=0 THEN PRINT :
     GOTO 2480
2520 IF A=1 THEN TYPE$="NO" ELSE TYPE$="CODE"
2530 PRINT
2540 PRINT "PLEASE INPUT KUTEN ";TYPE$;" :"
2550 PRINT " [ku],[ten] ﾄｼﾃ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ "
2560 INPUT "             : ";KUTEN1,KUTEN2
2570 IF (KUTEN1=0)+(KUTEN2=0) THEN 2540
2580***** GET PLACE FOR KANJI DATA
2590 GOSUB 2890
2600 HDATA=0
2610 IF (KUTEN1>=RECDATA(1))*(KUTEN1=<RECDATA(2)) THEN HDATA=0:
     GOTO 2660
2620 IF (KUTEN1>=RECDATA(3))*(KUTEN1=<RECDATA(4)) THEN HDATA=752:
     GOTO 2660
2630 IF (KUTEN1>=RECDATA(5))*(KUTEN1=<RECDATA(6)) THEN HDATA=4136:
     GOTO 2660
2640 PRINT "KUTEN ";TYPE$;" ｶﾞ ﾁｶﾞｲﾏｽ!!":
     GOTO 2540
2650****  Search place for write kanji font.
2660 SEARCHNO=(KUTEN1-1)*94+KUTEN2-HDATA
2670 IF TYPE$="CODE" THEN SEARCHNO=(KUTEN1-33)*94+(KUTEN2-32)-HDATA
2680 OPEN #"FD1","@KJPAT",1,"I/O"
2690 SEARCH #1,"SR",SEARCHNO
2700 FORMAT$="%"+STRING$(30," ")+"%"
2710 WRITE #1 USING FORMAT$;KNJPAT$
2720 CLOSE #1
2730 RETURN
2740*
2750* DYSPLAY FONT
2760*
2770 IF TYPE$="NO" THEN 2800
2780  KUTEN1=KUTEN1-$20
2790  KUTEN2=KUTEN2-$20
2800 W1$="0"+STR$(KUTEN1)
2810 W1$=RIGHT$(W1$,2)
2820 W2$="0"+STR$(KUTEN2)
2830 W2$=RIGHT$(W2$,2)
2840 W1$=W1$+W2$
2850 KNJ&=KANJI&(W1$)
2860 PRINT "RECORD NO=";SEARCHNO;"   ";KNJ&
2870 PRINT
2880 RETURN
2890*
2900* READ DATA FOR CHECK KUTEN
2910*
2920 RESTORE
2930 DIM RECDATA(6)
2940 FOR I=1 TO 6
2950   READ RECDATA(I)
2960 NEXT I
2970 DATA 1,7,16,47,84,94
2980 IF TYPE$="NO" THEN GOTO 3030
```

```
2990 FOR I=1 TO 6
3000   READ RECDATA(I)
3010 NEXT I
3020 DATA 33,39,48,79,116,126
3030 RETURN
```

漢字入出力のユーティリティとしてカナ漢字パッケージが東芝より発売されています．この内容は漢字入力サブルーチンと漢字パターン登録ユーティリティです．特に漢字パターン登録ユーティリティを使用することにより外字の登録は容易になります．

第8章

MINI-PASCAL

第8章　MINI-PASCAL

8-1　概略

8-1-1　PASCAL(パスカル)とは?

この章では，PASOPIA用ROMPAC，MINI-PASCALについて説明しますが，その前にPASCALとはいったいどんなものか，という点について少し説明しておきましょう．

PASCALというのは，BASIC，FORTRANなどと同じく，プログラミング言語の名前です．今までBASICしか使ったことのない方には，わかりにくいかもしれませんが，人間がコンピュータに自分の意志を伝えるには，ある定まったスタイルに従わなくてはなりません．その形式がプログラミング言語と呼ばれるものですが，その中でも特に人間にわかりやすく，記述しやすい言語は高級言語と呼ばれ，BASIC，FORTRAN，そしてこれから話をするPASCALなどの言語がその代表とされています．

普通，パーソナルコンピュータには，はじめから初心者でも使えるように，そのためのプログラム言語(BASICインタプリタ)が組み込まれているのですが，これはあくまでもプログラムですから，それを入れ換えれば，他のプログラミング言語も使えるようになるわけです．パソピアの場合，ROMPACの差し換えだけで，それが可能になっています．

それでは，PASCALの特色について説明しましょう．

PASCALは，1970年，N. Wirthによって開発されたものであり，その名前は，フランスの数学者，物理学者，哲学者であるブレーズ・パスカルにちなんでつけられました．これは，彼が計算機の初期の開発者であるためです．(パスカルは，実用可能な計算機を製作した一人と言われています．もちろん歯車式ですが)．

このPASCALの設計思想は，一言で言えば，構造化プログラミングを可能にしたことです．

これをわかっていただくために，BASICとの対比をしてみましょう．まず，BASICの複雑なプログラムを思い出して下さい．いきあたりばったりに大きなプログラムをBASICで書いた結果，自分でも何だかわからなくなってしまった経験をお持ちの方も少なくないことでしょう．また，他人の書いたプログラムを解読するときに，非常な努力がいるということも言えます．この原因としては，次のようなことが考えられます．

① GOTO文であちこちとびまわることができるため，プログラムの流れが追いにくい．

② FOR～NEXT以外のループは，GOTO文によって脱出するため，どこからどこまでがループなのかがわかりにくく，特に多重ループになるとますます混乱する．

③ 変数名が勝手に使えるため，型の混乱を生じやすい．

④ 複数の文をつなげるには，単にマルチステートメントにして並べるだけのため，文のまとまりがわかりにくい．

⑤ データが相互に関連を持った形で扱えず，むやみに配列に頼るため，データのまとまりが見えにくくなる．

このように，BASICには，プログラムの流れをさまたげる，多くの欠点があります．対してPASCALはどうでしょう．

上の5つに対応するPASCALの特色をあげると，次のようになります．

① GOTO文は，普通使用しない．(GOTO文のないプログラミングイコール構造化プログラミングと言われることさえある)．

② ループの表現には，FOR～NEXTのほか，WHILE～DO，REPEAT～UNTILがあり，頭からプログラムを読んでいってもループの構造がはっきりわかる．

③ 変数は，その名前，型をすべて前もって宣言しなければならず，また，その有効範囲もはっきりしておりわかりやすい．

④ 複数の文をまとめるBEGIN～ENDがあり，まとまりが明確になる．

⑤ 豊富なデータ型を扱うことができる．

このように，PASCALによるプログラミングでは，プログラムのすっきりとした直線的構造とデータの相互に結びついた構造を実現することが可能になっています．このような，プログラム手法を構造化プログラミングと呼ぶのです．

PASCALには，BASICなどのインタプリタと違って，はじめにプログラムを機械語などに翻訳してしまい，まとめて実行するコンパイラという方式をとっているものや，Pコードと呼ばれるコードに翻訳して，実行の際は，Pコードインタプリタでそれを解釈する方式，ハード的に直接Pコードを実行する方式(パスカル・マイクロエンジン等)などいくつかの型式がありますが，先ほど述べた構造化プログラミングはどのタイプでも関係なく可能です．

以上，概略を説明してきましたが，PASCALは，そのわかりやすさと強力さのため，現在世界中で急速に普及しつつあり，FORTRANやBASICにかわって，次の時代の中心となるプログラミング言語として，大きな期待がかけられています．PASCALを知ることは，プログラムの設計や解釈に十分役立つことと思います．

8-1-2　MINI-PASCALの位置付け

8-1-1では，PASCALの概要について説明してきたわけですが，PASOPIA用MINI-PASCALについてはどうでしょうか．

同じBASICでも機種によってさまざまな違いがあるように，PASCALでもいくらか違いがあります．ここで述べるPASOPIA用MINI-PASCALは，標準的PASCALとは違い，PASCALの基本的なしくみをマスターするための教育用システムと言えるものです．

まず，標準的なPASCALに比べ以下の点が縮小されています．

① LABELが付けられない．
② 定数，型(CONST，TYPE)が定義できない．
③ 変数の型は，整数型，文字型のみで，配列は整数型しかとれない．そのため，他の型を必要とする文や関数等はない．
④ CASE文がない．
⑤ インタプリタである．

また，逆にPASOPIAの能力を生かすために，グラフィック関係の命令やPEEK, POKE, CALL, ADR, INP, TIMEなどが追加されています．したがって，一般にPASCAL用として発表されているプログラムを走らせる際は注意が必要です．

このように，機能的にある程度制限されてはいますが，低価格のパーソナル・コンピュータ用にディスク不用で手軽なROMPACとしてPASCALが発売されたということは，大いに注目すべきことでこのMINI-PASCALは，PASCALの特徴的な記述スタイルを身に付けるには十分な機能を持っています．

今までBASICしか使ったことのなかった方々も，これによってPASCALの世界に目を向けて下さることと思います．

8-1-3　プログラムの構成

では，具体的にMINI-PASCALのプログラムについて説明することにしましょう．MINI-PASCALでは，BASICのように命令を直接実行(行番号を付けずに実行)することはできません．それに，プログラムの書き方も最初はわかりにくい点がありますので，しっかり覚えておいて下さい．では，まず次のプログラムを見て下さい．

```
1      PROGRAM SAMPLE;
2      VAR A,B,C,D:INTEGER;
3      BEGIN
4        WRITE('input a,b=? ');
5        READ(A,B);
6        C:=A*B;
7        D:=A DIV B;
```

```
8       WRITE('a*b=',C,/);
9       WRITE('a/b=',D,/)
10   END.
```

このプログラムは，単に説明のためにつくったもので，大して意味はありませんが，MINI-PASCALのプログラム構成はよくわかると思います．

まず，プログラムはPROGRAMではじまり〝.〞（ピリオド）で終わらなくてはなりません．そして，VARではじまる変数宣言部があり，その後に手続き宣言部，関数宣言部が続き，実行部はBEGINではじまりENDで終わります．

変数宣言部というのは，プログラム中で使う変数名，型を前もって宣言しておくもので，BASICと違い勝手に変数を使うわけにはいきません．これは，変数を全く使わないプログラムであれば不要です．

手続き宣言部，関数宣言部というのは，BASICのサブルーチンや関数の定義などにあたるもので，これも前もって宣言しておかなくてはなりません．そして，それぞれVARではじまる変数宣言部，BEGINからENDまでの実行部で成り立っています．これらも，使わないときは宣言しなくてもかまいません．また，複数の手続きや関数を宣言することもできます．

次に，前出のプログラムを見て〝;〞が多く使われている点に注目して下さい．これは，文の区切りであり，BASICの〝:〞と似ています．しかし，どういうところで付けなければならないか，どういうところは付けなくてよいか，という点で混乱しがちです．実際，初期のマニュアルのプログラム例は混乱していますので，次にまとめておきましょう．

まず，プログラム頭部，変数宣言部，手続き宣言部，関数宣言部，プログラム実行部を区切るために〝;〞が必要です．また，文と文の間も〝;〞で区切ります．しかし，ここで文とは一体何を意味するのかですが，正確な定義は8-2-1の構文図を見ていただくことにして，ここでは簡単に説明しておきましょう．

① 代入文
② 手続きの呼び出し
③ IF～THEN～(ELSE～)
④ WHILE～DO～
⑤ REPEAT～UNTIL～
⑥ FOR～TO(DOWNTO)～DO～
⑦ BEGINとENDが〝;〞で区切られた文をはさんだもの

以上のものを文と考えて下さい．ただし定義⑦の中で，文という言葉がありましたが，これは循環論法的定義ではなく，いくつかの文があったとき，それらを〝;〞で区切り，BEGINとENDではさむとあらたに文ができるということです．ここで注意することは，BEGIN，END，その他の予約語は文ではないということです．つまり，BEGINとENDの間に文が4つあるとすれば，必

要な "；" は３つで，BEGINの直後，ENDの直前には不要になります．

また，③～⑥の定義の中で～と書いたところにも，いくつか文の入る場所があります．定義⑤のREPEATとUNTILの間（REPEATの後，UNTILの前には "；" は不要です）に，複数の文を "；"で区切って入れることができます．定義③，④，⑥のELSE，THEN，DOの後には，１つの文を入れることができます．こういう場所に複数の文を入れたいときは，BEGINとENDではさみ "；" で区切って１つの文とすることができます．

以上，説明の例として次のプログラムをあげておきますので，"；"の入る位置，入らない位置をマスターして下さい．

"；"　の必要な場所、不要な場所

```
1    PROGRAM SAMPLE;              プログラム頭部の終わりなので必要
2    VAR A,B,C,D,E:INTEGER;       グローバル変数宣言部の終わりなので必要
3    BEGIN                        BEGINは文ではないので不要
4      A:=10;                   ┐
5      B:=-5;                   │ 次の文との区切りに必要
6      C:=A+B;                  │ （次のREPEAT～UNTIL A < 0 を一つの文と考える．）
7      D:=2*A+B DIV 2;          ┘
8      REPEAT                     REPEAT は文ではないので不要
9        A:=A-1;                ┐ 次の文との区切りに必要
10       WRITE('%');            ┘
11       B:=B+1                   次は、文ではないので不要
12     UNTIL A<0;                 次の文 （WHILE ～18行め） との区切りに必要
13     WHILE B>0 DO               DOのあとはまだ文の続きなので不要
14       BEGIN                    BEGINは文ではないので不要
15         B:=B+D-C;            ┐ 次の文との区切りに必要
16         D:=D DIV 2;          ┘
17         WRITE('$')             次のENDは文ではないので不要
18       END;                     次の文 （FOR ～23行め） との区切りに必要
19     FOR D:=1 TO C DO           DOのあとはまだ文の続きなので不要
20       BEGIN                    BEGINは文ではないので不要
21         WRITE('#');            次の文との区切りに必要
22         WRITE('*')           ┐ 次のENDは文ではないので不要
23       END                    ┘
24   END.                         最後なので不要．（"・"が必要）
```

さらにもう１つ "；" について説明を加えておきましょう．BASICでは，行の終わりには単にRETキーを押すだけで，"："を付ける必要はありませんでした．しかし，今までの例でわかるように行の終わりにも "；" を付けて改行しているところがたくさんあります．これは，PASCALでは行の区切りというのは，単にプログラム編集上の目安でしかなく "；" があれば区切り，なければスペースとみなすからです（ただし，コメント，' で囲んだ文字列などの場合は別です）．したがって "；" を忘れると，その次の行とつながっていると解釈されてしまいます．逆に１つの行でも "；" で区切れば，複数の文を書くことができます．

以上の説明の例として，次の３つのプログラムを比べてみて下さい．これらは，いずれも同じ働きをするプログラムです．

```
1    PROGRAM SAMPLE1;
2    VAR X,Y,Z:INTEGER;
3    BEGIN
4      WRITE('x=? ');
5       READ(X);
```

```
6       WRITE('y=? ');
7       READ(Y);
8       Z:=X*X+Y*Y;
9       WRITE('x*x+y*y =',Z)
10    END.
```

```
1     PROGRAM SAMPLE2;
2     VAR
3         X,Y,Z:INTEGER;
4     BEGIN
5       WRITE
6             ('x=? ');
7       READ(X)
8       ;
9       WRITE
10      ('y=? ');
11      READ(Y);
12      Z:=
13      X*X
14      +Y*Y
15      ;
16      WRITE
17      ('x*x+y*y = '
18      ,
19      Z)
20    END.
```

```
1     PROGRAM SAMPLE3; VAR X,Y,Z:INTEGER; BEGIN WRITE('x=? '); READ(X); WRITE('y=
? ');
2     READ(Y); Z:=X*X+Y*Y; WRITE('x*x+y*y =',Z) END.
```

以上でわかるように，〝；〞は区切りをはっきりさせる印です．必要なところで忘れないように十分注意して下さい．

最後に，インデント（プログラムの段付け）について説明しておきましょう．この章やMINI-PASCAL付属のマニュアル，一般のPASCAL解説書などのプログラム例では，通常，BEGIN，END等ではさまれた文を右にずらして段が付けられています．これは，プログラムの構造を見やすくするために付けられたもので，文法的には，はっきり決まっているわけではありませんが，適切なインデントを行う習慣を付けておくことが望ましいでしょう．

例えば次の3つのプログラムは，いずれも同じ働きをしますが，どれが一番わかりやすいかは言うまでもないでしょう．

```
1     PROGRAM SAMPLE1;
2     VAR INPUT,SUMSQR,COUNTER:INTEGER;
3     BEGIN
4       WRITE('input N',/,'?');
5       READ(INPUT);
6       SUMSQR:=0;
7       COUNTER:=INPUT;
8       WHILE COUNTER>0 DO
9         BEGIN
10          SUMSQR:=SUMSQR+COUNTER*COUNTER;
11          COUNTER:=COUNTER-1
12        END;
13      WRITE('"SUM of 1*1,2*2,...,N*N"= ',SUMSQR,/)
14    END.
```

```
1      PROGRAM SAMPLE2;
2      VAR INPUT,SUMSQR,COUNTER:INTEGER;
3      BEGIN
4      WRITE('input N',/,'?');
5      READ(INPUT);
6      SUMSQR:=0;
7      COUNTER:=INPUT;
8      WHILE COUNTER>0 DO
9      BEGIN
10     SUMSQR:=SUMSQR+COUNTER*COUNTER;
11     COUNTER:=COUNTER-1
12     END;
13     WRITE('"SUM of 1*1,2*2,...,N*N"= ',SUMSQR,/)
14     END.
```

```
1         PROGRAM SAMPLE3;
2      VAR INPUT,SUMSQR,COUNTER:INTEGER;
3              BEGIN
4       WRITE('input N',/,'?');
5                 READ(INPUT);
6         SUMSQR:=0;
7      COUNTER:=INPUT;
8                    WHILE COUNTER>0 DO
9             BEGIN
10       SUMSQR:=SUMSQR+COUNTER*COUNTER;
11                              COUNTER:=COUNTER-1
12            END;
13              WRITE('"SUM of 1*1,2*2,...,N*N"= ',SUMSQR,/)
14          END.
```

8-1-4　エディタの使い方など

ここでは，本に載っているプログラムや自分で考えたプログラムの入力方法，また入力したプログラムの変更法や保存法について説明します．

MINI-PASCALで使用される文字は，英字(A～Z)，数字(0～9)，特殊記号(+, −, ∗, /, :=, ., ..., ;, :, =, 〈〉, <=, >=, >, <, (,), [,], (∗, ∗), #)です．ただし，コメントや“'”ではさまれた文字列中には，これ以外の文字を使ってもかまいません．

本に載っているPASCALのプログラムは，しばしば英小文字で書かれており，BEGINなどが太字になっていますが，ここではすべて英大文字を使います(入力するときは，小文字を使ってもかまいませんが，LISTをとると大文字になっています)．

BASICと異なる使い方をする特殊記号については，:=は代入，[,]は配列，(∗, ∗)はコメントとして用います．また，2文字からなる特殊記号の間にスペースを入れたり，<>, <=, >=をそれぞれ><, =<, =>としたりしてはいけません．

BASICと違うキーの使い方について，次にまとめておきましょう．

① ↑, ↓, ←, →, DEL, INS, スペース以外のキーは，押したままにしてもリピートしません．

② CTRLキーとSTOPキーは，プログラムの実行を中断させるため，同時に押したとき以外は無効です．

③ ESCキーは入力の際，BASICのSTOPキーと同様の効果があり，その入力をキャンセルして改行します．

④　ファンクションキーは無効です．

⑤　↑，↓のキーは，EDIT時は入力を受け付け，上下の行を表示しますが，INSERT時はRETキーと同じ働きをします．

⑥　CLS，HOME，LABEL，TABなどのキーは無効です．

⑦　←，→のキーは，すでに文字を入力した範囲のみ有効で，左右の端までいくとそれ以上はききません．

⑧　DELキーのきき方はBASICと違い，カーソルのある位置の文字が抹消され，それより右の文字列は一文字ずつ左へずれます．

以上のことに注意して，MINI-PASCALのプログラム入力をして下さい．

次に，エディタの各命令についてですが，くわしくは付属のマニュアルに譲り，ここでは注意すべき点だけをあげておきましょう．

まず，ランダムな文字列をプログラムとして入力してみて下さい．〝?〟が出て再入力を促されるときとそうでないときがあると思います．これは，無意味な文字列は，すべて名前とみなして解釈するため，名前とみなせないような入力に対しては〝?〟を出して再入力を要求するからです．つまり，かな文字やグラフィックキャラクタ等を含む場合，9文字以上の長さの場合，5Aなどのように名前として判断されない並びの場合です．また，〝′〟で囲まれた文字列の場合は必ず〝′〟を偶数個必要とするのに，奇数個しかなかった場合にも〝?〟が出て再入力を要求します．その他，もし正しくない文が受け付けられた場合でも，実行すれば当然エラーとなってしまいます．さらに，EDITの際一番上の行で〝↑〟を入力したり，一番下の行で〝↓〟を入力した場合もエラーになります．DELETE　m，nは「m行からn行までを削除」の意味であるのに，LIST　m，nは「m行からn行分を表示」の意味になることにも注意して下さい．

最後に，プログラム中のスペースの扱いについて説明しておきます．各行のインデントのためのスペースは，LISTの際入力した通りに表示されますが，その他のスペースで不要なものは，すべて詰められてしまいます．必要なスペースとは，各予約語の前後1文字だけですが，これも，予約語の前後に〝)〟などの特殊記号がある場合には詰められてしまいます．つまり，インデント以外のスペースは，入力しても意味がないということです．

以上，エディタの一般的使い方にはあまり触れませんでしたが，後はマニュアルをよく読み〝ポインタ〟の考えをしっかりつかんでエディタを使いこなして下さい．

8-2　MINI-PASCALのしくみ

8-2-1　構文図

ここでは，MINI-PASCALの文法などの細かい点について話を進めていきましょう．まず，構文図によって文法をはっきりさせてみましょう．図8-2-1を見て下さい．

・因子

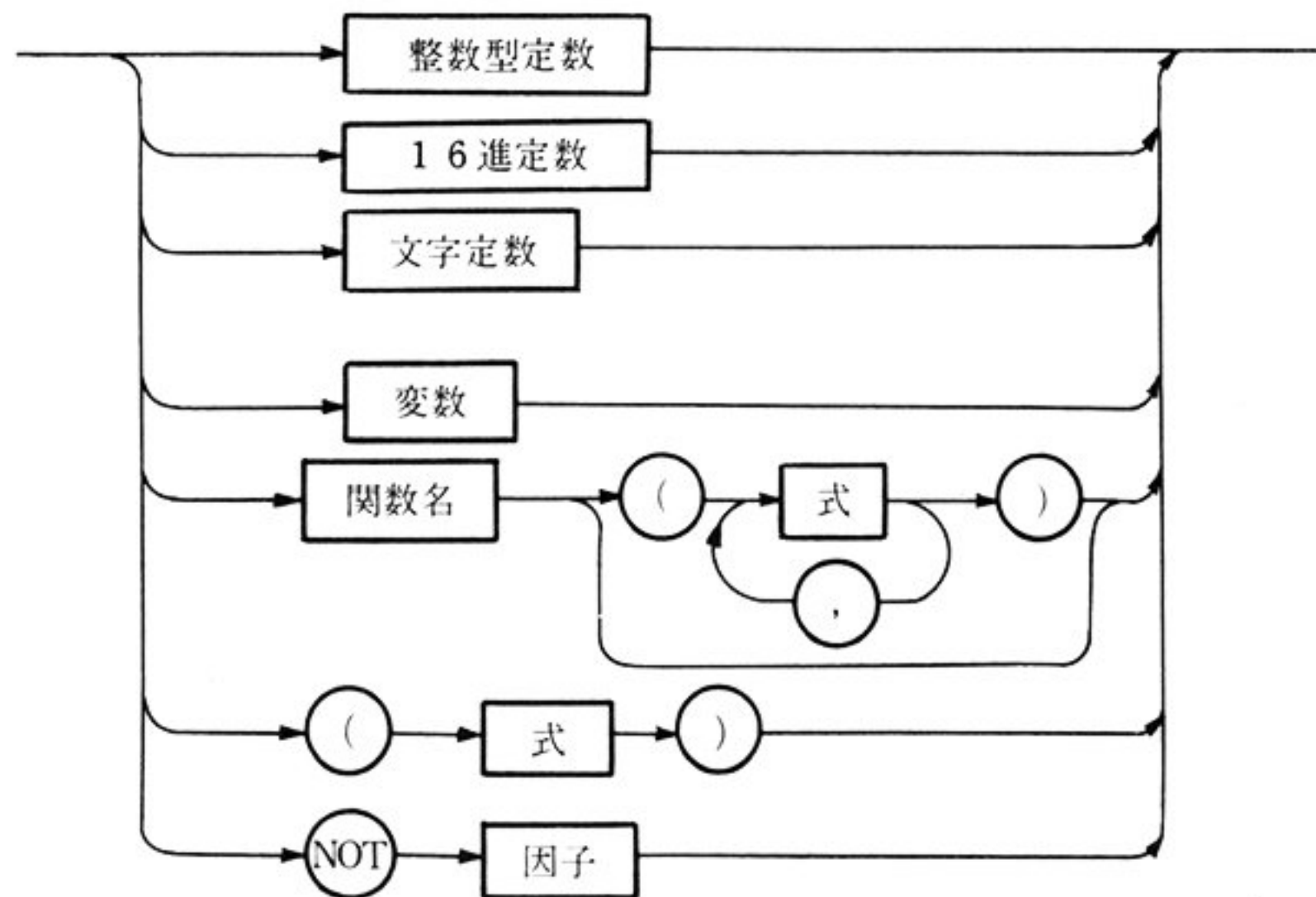



図8－2－1 構文図（1）

これが構文図というもので，この図は因子はどういう働きをするかを示しています．枝分かれしている矢印は「または」を表します．すなわち，この図は因子とは，

① 整数型定数
② 16進数定数
③ 文字定数
④ 変数
⑤ 関数名，あるいはその後に "(", ")" ではさまれ "," で区切られた式が1つ以上並んだもの
⑥ "(", ")" で式をはさんだもの
⑦ NOTと因子を並べたもの

のいずれかである，ということを意味しています．⑤については矢印がループになっているところが「式」または「式，式」または「式，式，式」または，「式，式，式，式，……」という意味になるので，結局，式を1つ以上並べて間を "," で区切ったもの，ということになります．また，⑦については因子にNOTを付けると，それもまた因子である，ということを示しているのです．

では，以下に他の構文図をあげていきましょう．なお，矢印がループになっているところで，そのループのまわる回数に限度のあるものについては，そのつど説明を加えます．

まずは，名前，整数型定数，16進定数，変数，文字定数です．名前は8文字以内なので，ループをまわる回数は0～7です．整数型定数は0～32767の範囲，16進定数は4桁以下となります．変数名の後に[,]が付くのは配列の場合で，MINI-PASCALでは一次元配列しかないため式は1つだけです．文字定数の文字とは，キーボードから打ち込めるあらゆる文字のことです．ここで，

“’” 自身を入れる場合は2つ並べて “’’” とします.

・**名前**

英字 英字 数字

・**整数型定数**

数字

・**16進定数**

数字 AからFの英字

・**変数**

変数名 [式]

・**文字定数**

' 文字 '

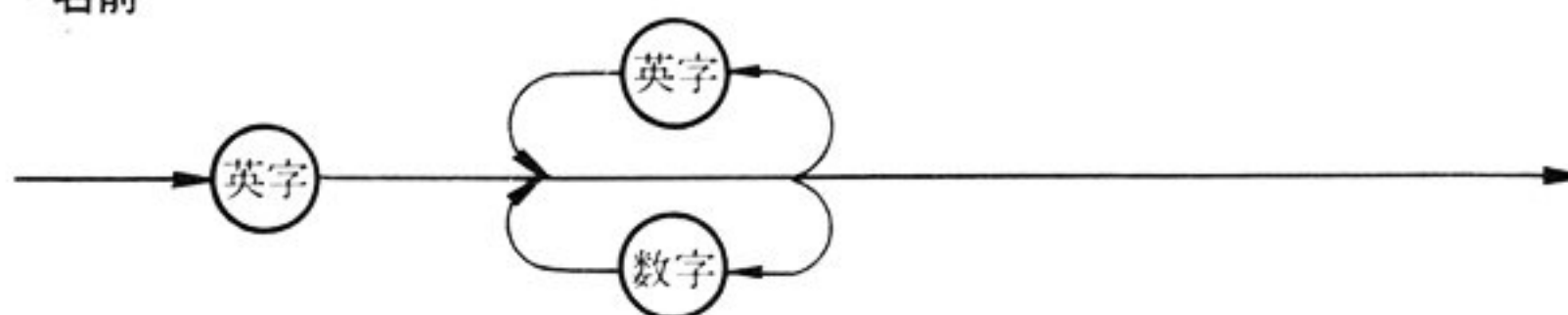
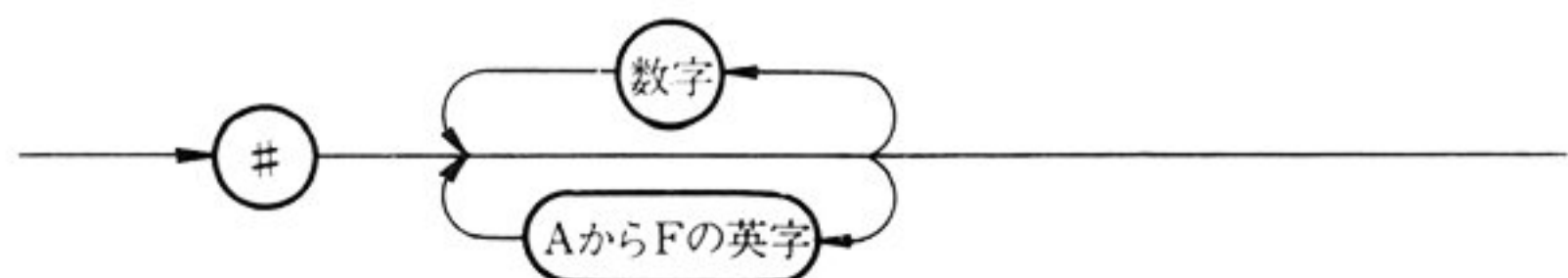
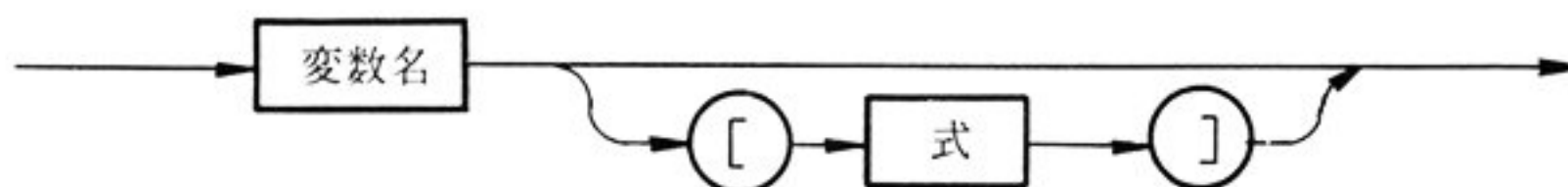
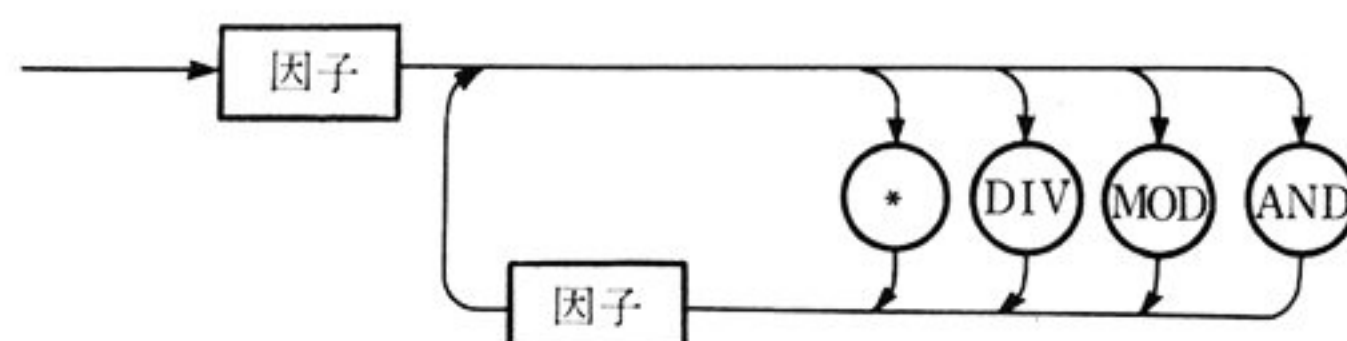
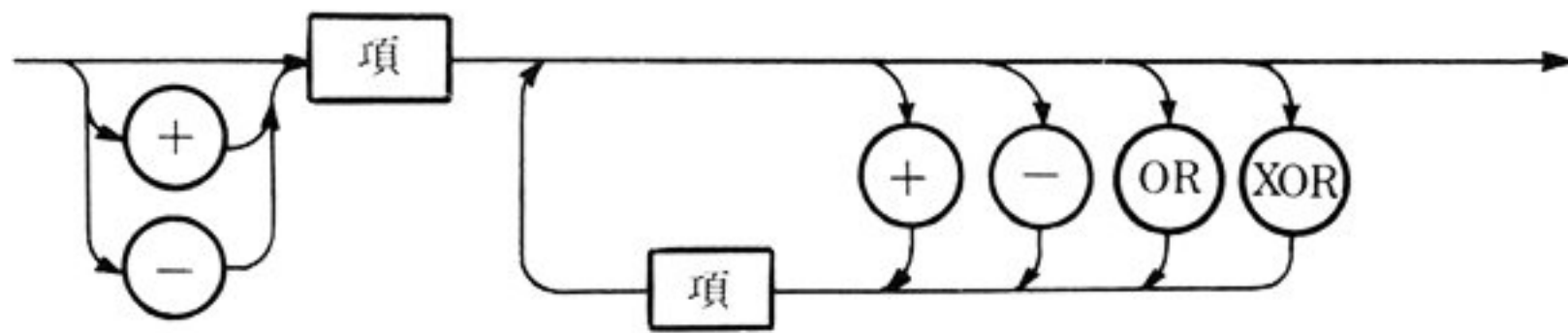
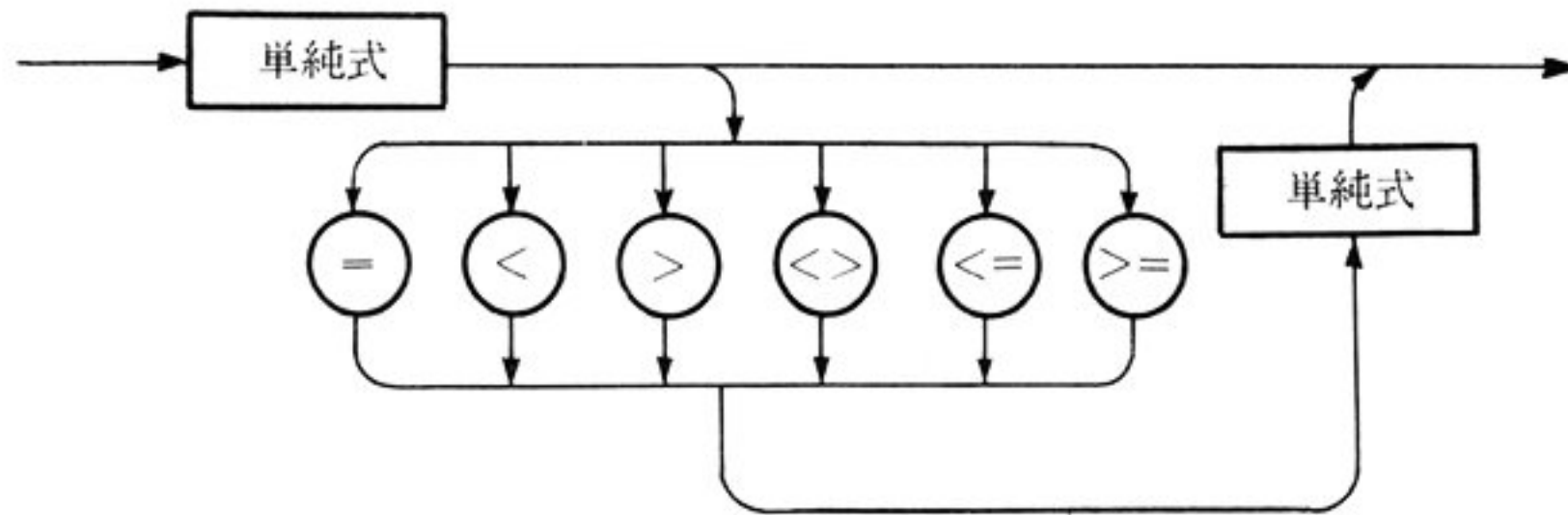



図8―2―2 構文図（2）

以下は，項，単純式，式，引数の並びの構文図です．＊，DIV，MOD，ANDは，＋，－，OR，XORより優先度が高いため，前者で結ばれているものは項，後者で結ばれているものは単純式となります．単純式またはそれらを＞，＝などでつないだものは式と呼ばれ，IF，WHILE，UNTILの後などに来ます．また，型名というのは，MINI-PASCALではINTEGER，STRINGのどちらかだけです．

・**項**

・単純式

・式

・引数の並び

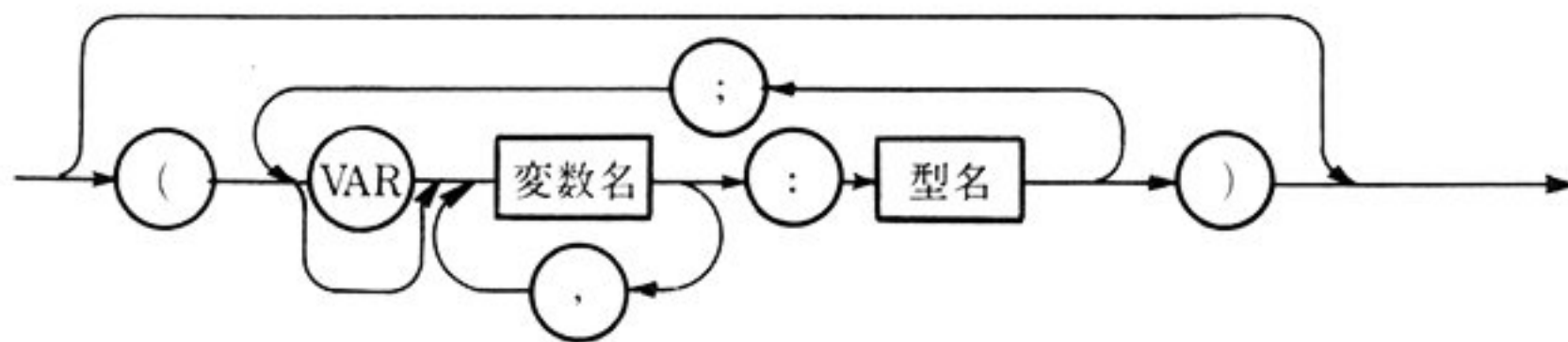



図8－2－3　構文図（3）

これは，ブロックの構文図です．PROCEDURE，FUNCTIONなどに用いられます．

・ブロック

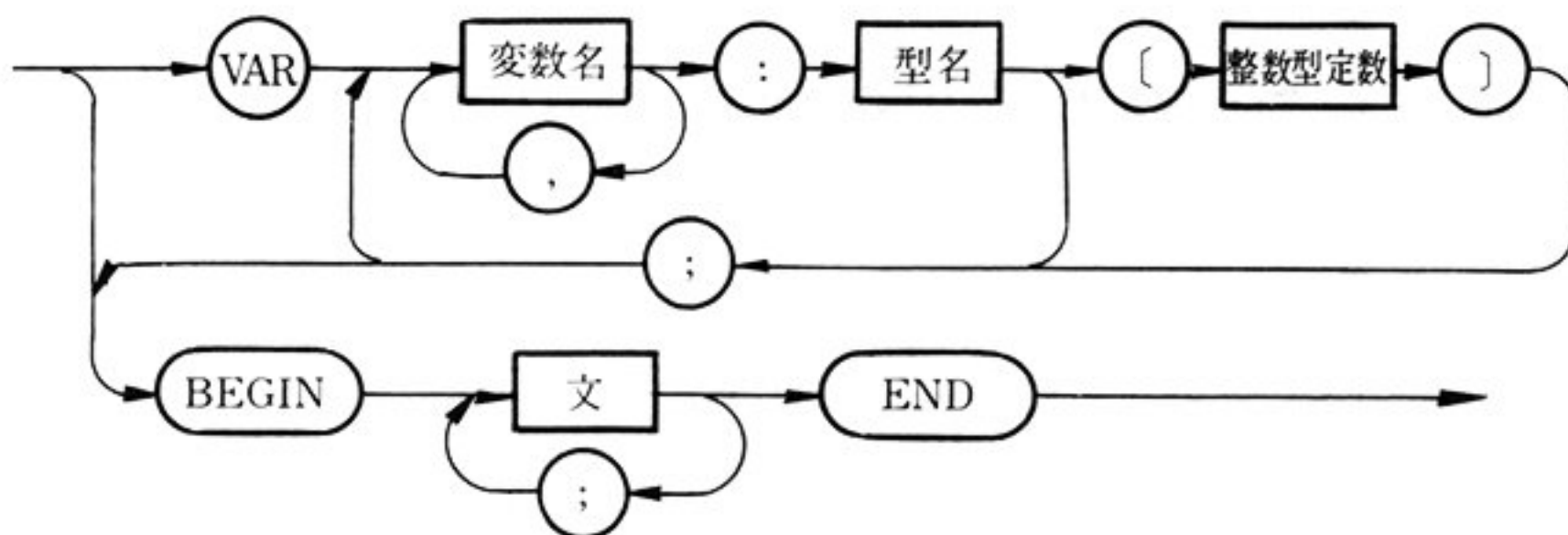



図8－2－4　構文図（4）

今度は文の構文図です．8－1－3で簡単に定義を述べましたが，これが正確な定義です．このうち，一番下の何も通っていない矢印は空文と呼ばれるものに対応します．例えば〝；〞を2つ並べると，その間には空文があるとみなされます．また，THEN，ELSE，DOの後には1つの文しか書けないことに注意して下さい．

・文

図8―2―5　構文図（5）

・プログラム

PROGRAM
名前
;
VAR
変数名
,
:
型名
〔
整数型定数
〕
;
;
ブロック
PROCEDURE
FUNCTION
名前
引数の並び
;
BEGIN
文
;
END
.

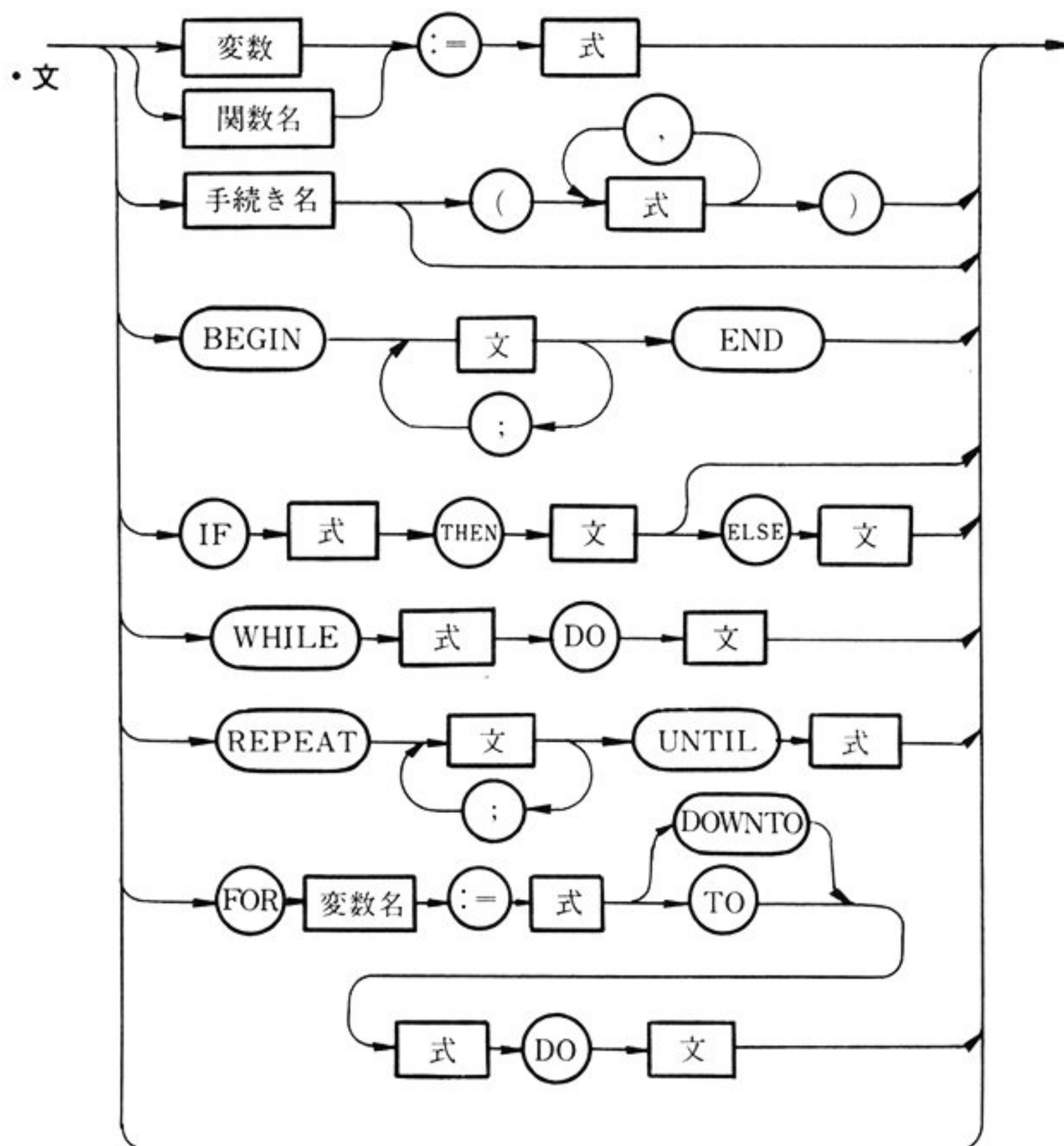
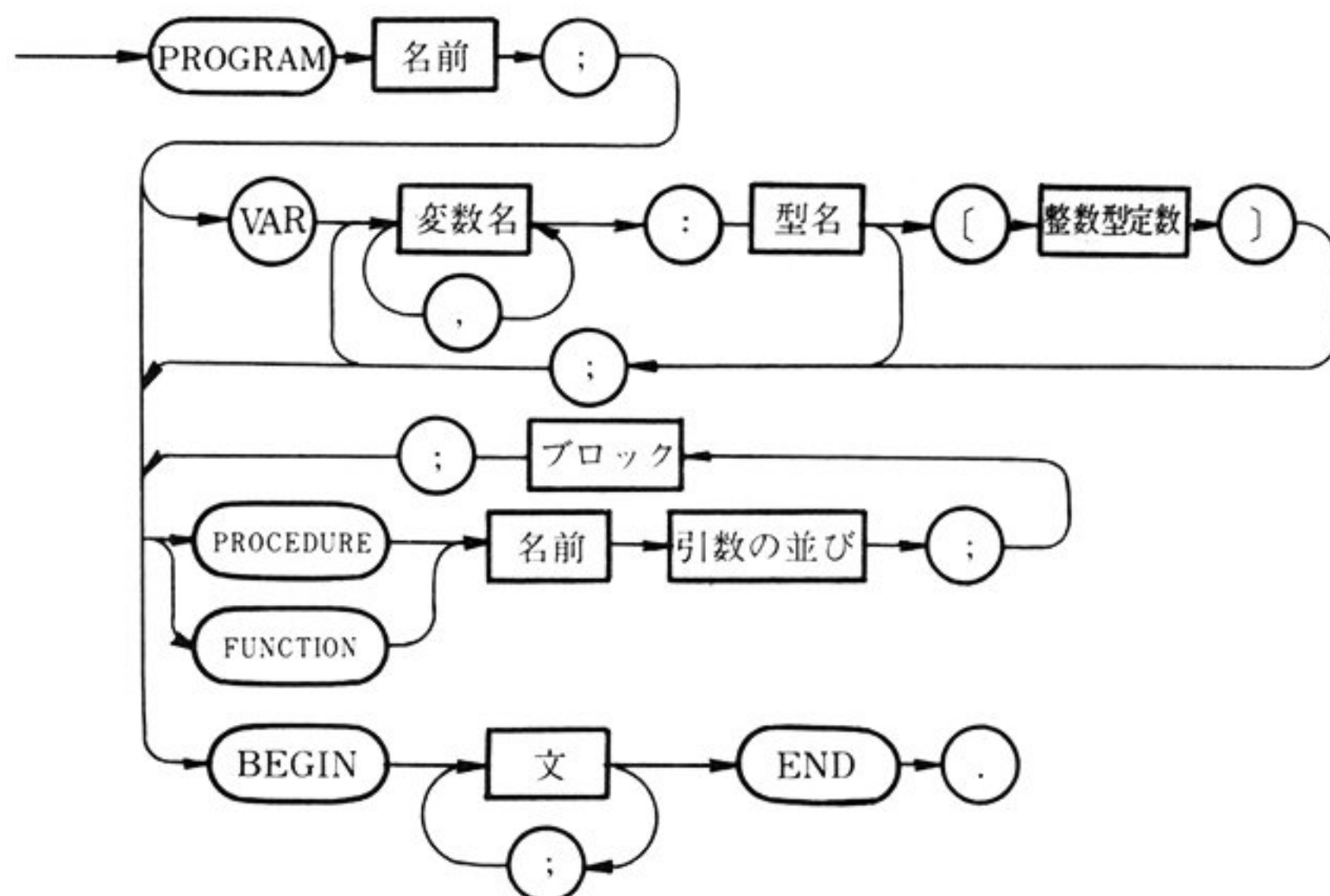



図8―2―6　構文図（6）

最後は，プログラムの構文図です．こういう形で表すと，プログラムとは何かがはっきりわかると思います．

8-2-2　変数について

MINI-PASCALの命令1つ1つについて説明を加えていくことはできませんので，マニュアルにはっきり書かれていないところ，わかりにくいところ，面白い使い方等に絞って説明することにします．まずは，変数についての話です．

MINI-PASCALで許される変数名は英字ではじまり，英数字からなる8文字以下の文字列で，使用する変数名は前もってVARの後で宣言しなければなりません．しかし，変数を宣言する場所はプログラム内に1か所だけではなく，PROCEDURE，FUNCTIONの中にもあります．では，これらの間の関係はどうなっているのでしょうか？　例えば，それぞれの中で同一の変数名を宣言して使うとどうなるのでしょうか．

BASICでは，変数の宣言をしないので，プログラム中のどこでも変数名は同じ値を持っていますが，MINI-PASCALではそうはいきません．プログラムの最初の部分(PROCEDUREやFUNCTIONの前)で宣言された変数をグローバル(大域)変数，PROCEDUREやFUNCTIONの中で宣言された変数をローカル(局所)変数と呼びます．ローカル変数は，そのPROCEDUREまたは，FUNCTIONの中でのみ有効です．それに対し，グローバル変数は原則的にはプログラム全体で有効ですが，PROCEDURE，FUNCTIONの中で同じ変数名が宣言されている場合は，そのPROCEDURE，FUNCTION内ではローカル変数の方が優先で，その変数名はPROCEDURE，FUNCTIONの中で代入された値を持つことになります．しかし，PROCEDURE，FUNCTIONから戻って来ると，再びグローバル変数として扱われ，PROCEDURE，FUNCTIONの内部で代入された値は捨てられ，呼び出す前の値が回復されます．例えば，次のプログラムの実行結果を見て下さい．

```
1     PROGRAM SAMPLE;
2     VAR I,J:INTEGER;
3     PROCEDURE TEST;
4       VAR I:INTEGER;
5       BEGIN
6         I:=100;
7         J:=200;
8         WRITE('...in procedure',/,'i = ',I,/,'j = ',J,/)
9       END;
10    BEGIN
11      I:=10;
12      J:=20;
13      WRITE('...before procedure',/,'i = ',I,/,'j = ',J,/);
14      TEST;
15      WRITE('...after procedure',/,'i = ',I,/,'j = ',J,/)
16    END.
```

PROCEDURE，FUNCTIONの中から，さらにPROCEDURE，FUNCTIONを呼び出しているときも，今と同様同じ変数名であらたに定義されていれば，ローカル変数が有効となり，そうでなければグローバル変数が有効となります．また，はじめの方のPROCEDURE，FUNCTIONで宣言されたローカル変数は，他のPROCEDURE，FUNCTIONを呼び出したときは無効となりま

す．

その例として，次のプログラムの実行結果を見て下さい．

```
1     PROGRAM SAMPLE;
2     VAR I,J:INTEGER;
3     PROCEDURE TEST1;
4       VAR I:INTEGER;
5       BEGIN
6         I:=100;
7         J:=200;
8         WRITE('...in procedure1',/,'i = ',I,/,'j = ',J,/);
9         TEST2;
10        WRITE('...after procedure2',/,'i = ',I,/,'j = ',J,/)
11      END;
12    PROCEDURE TEST2;
13      VAR J:INTEGER;
14      BEGIN
15        I:=1000;
16        J:=2000;
17        WRITE('...in procedure2',/,'i = ',I,/,'j = ',J,/)
18      END;
19    BEGIN
20      I:=10;
21      J:=20;
22      WRITE('...before procedure1',/,'i = ',I,/,'j = ',J,/);
23      TEST1;
24      WRITE('...after procedure1',/,'i = ',I,/,'j = ',J,/)
25    END.
```

さらに，PROCEDURE，FUNCTIONには，変数の値を引き渡すことができます．特にFUNCTIONのときは，これができなくてはどうにもなりません．PROCEDURE，FUNCTIONの宣言の際，名前に続けて“(”と“)”で変数の並びをはさんでおき，呼び出す際にPROCEDUREやFUNCTIONの名前の後に“(”と“)”で値を並べたものをはさむ，ということです．例えば，PROCEDURE SAMPLE(A，B：INTEGER)；と宣言しておき，呼び出す際にSAMPLE(1，2)とすれば，A，Bにそれぞれ1，2という値が入ります．このとき，A，BはPROCEDURE内部で定義されたローカル変数と同じ扱いになり，プログラムの実行部にグローバル変数A，Bがあっても無関係です．

次のプログラム例を見て下さい．呼び出す際にローカル変数と同じ名前を使っていても，グローバル変数の値は変化していないことがわかるでしょう．

```
1     PROGRAM SAMPLE;
2     VAR I,J:INTEGER;
3     PROCEDURE TEST(I:INTEGER);
4       BEGIN
5         I:=100;
6         J:=200;
7         WRITE('...in procedure',/,'i = ',I,/,'j = ',J,/)
8       END;
9     BEGIN
10      I:=10;
11      J:=20;
12      WRITE('...before procedure',/,'i = ',I,/,'j = ',J,/);
13      TEST(I);
14      WRITE('...after procedure',/,'i = ',I,/,'j = ',J,/)
15    END.
```

以上のように，ローカル変数に代入された値は，呼び出しから戻ってくるとすべて失われてしまうため，困ることがあります．何らかの計算結果を返してほしいという場合です．MINI-PASCALではFUNCTIONの値は単独の整数型の値しかとれないため，複数の数値や文字型の値を返してほしい場合に困ってしまいます．あらかじめ宣言しておいたグローバル変数に値を代入する，という方法が考えられますが，毎回同じ変数に値が入ってくるというのも不便です．PROCEDUREやFUNCTIONから直接値を戻すことはできないのでしょうか？

前にあげた図8-2-10を見直して下さい．変数名の前にVARというのがあります．これを使ってみましょう．例えば，PROCEDURE　SAMPLE(VAR　A：INTEGER)；と宣言しておいて，SAMPLE(B)と呼び出されたとき，AにBの値が代入されるのは前と同じですが，そこから先が違い，例えばPROCEDURE内で，A：＝A＊2としますと，戻ってきたときにはBの値は2倍になっています．すなわち，VARを伴って宣言された場合は，その変数の最終値が，呼び出したときの変数に返されるということです．したがって，この場合，呼び出す方は引数を変数にする必要があり，SAMPLE(1)のような呼び出しはできません．このときの引数を変数引数，前の例のときの引数を値引数と呼びます．

この方法をとれば，いくらでも値を返すことができます．また，宣言された変数名はローカル変数として扱われます．

以上のことを次の例で確認して下さい．

```
1     PROGRAM SAMPLE;
2     VAR I,J:INTEGER;
3     PROCEDURE TEST(I:INTEGER; VAR K:INTEGER);
4       BEGIN
5         I:=I*10;
6         K:=K*10;
7         WRITE('...in procedure',/,'i = ',I,/,'j = ',J,/,'k = ',K,/)
8       END;
9     BEGIN
10      I:=10;
11      J:=20;
12      WRITE('...before procedure',/,'i = ',I,/,'j = ',J,/);
13      TEST(I,J);
14      WRITE('...after procedure',/,'i = ',I,/,'j = ',J,/)
15    END.
```

この方法を使えば，文字列の操作も簡単にできます．MINI-PASCALについている文字列操作の手続きは，INSERT，DELETE，MOVEですが，これらを用いてBASICのRIGHT$，LEFT$，MID$に相当するものをつくってみましょう．

BASICで，

① A$＝RIGHT$(B$，I)

② A$＝LEFT$(B$，I)

③ A$＝MID$(B$，I，J)

に対応するものをここでは，それぞれ

① RIGHT(A，B，I)

② LEFT(A, B, I)

③ MID(A, B, I, J)

と書くことにします．プログラムは次のようになります．

```
1      PROGRAM SAMPLE;
2      VAR A,B:STRING;
3          I,J:INTEGER;
4      PROCEDURE RIGHT(VAR A,B:STRING; I:INTEGER);
5        VAR C:STRING;
6        BEGIN
7          C:=B;
8          DELETE(C,1,LENGTH(B)-I);
9          A:=C
10       END;
11     PROCEDURE LEFT(VAR A,B:STRING; I:INTEGER);
12       VAR C:STRING;
13       BEGIN
14         C:=B;
15         DELETE(C,I+1,LENGTH(B)-I);
16         A:=C
17       END;
18     PROCEDURE MID(VAR A,B:STRING; I,J:INTEGER);
19       VAR C:STRING;
20       BEGIN
21         C:=B;
22         DELETE(C,1,I-1);
23         DELETE(C,J+1,LENGTH(C)-J);
24         A:=C
25       END;
26     BEGIN
27       WRITE('right',/);
28       READ(B);
29       READ(I);
30       RIGHT(A,B,I);
31       WRITE(A,/);
32       WRITE('left',/);
33       READ(B);
34       READ(I);
35       LEFT(A,B,I);
36       WRITE(A,/);
37       WRITE('mid',/);
38       READ(B);
39       READ(I,J);
40       MID(A,B,I,J);
41       WRITE(A,/)
42     END.
```

以上の変数引数に関することは，マニュアルにはあまり詳しく書かれていませんが，大変役にたつテクニックだと思います．また，マニュアルにはFUNCTIONの例があまりあげられていませんので，ここで最大公約数を与えるユークリッドのアルゴリズムを関数の形で書いたものをあげておきましょう．(変数の関係に注意して下さい)．

```
1      PROGRAM SAMPLE;
2      VAR M,N:INTEGER;
3      FUNCTION GCM(M,N:INTEGER);
4        VAR L:INTEGER;
5        BEGIN
6          IF M<0 THEN M:=-M;
7          IF N<0 THEN N:=-N;
8          WHILE M>0 DO
9            BEGIN
```

```
10          IF M<N THEN BEGIN
11                       L:=M;
12                       M:=N;
13                       N:=L
14                     END;
15          M:=M MOD N
16        END;
17      GCM:=N
18    END;
19  BEGIN
20    WRITE('m,n=? ');
21    READ(M,N);
22    WRITE('(m,n) = ',GCM(M,N),/)
23  END.
```

8-2-3 再帰的呼び出し

今度は，再帰的呼び出し(recursive call)を取り上げましょう．

再帰的というのは，自分自身を呼び出すということを意味します．すなわち，PROCEDUREやFUNCTIONの中で，そのPROCEDUREやFUNCTION自身を使うということです．

その例としては，階乗の計算がよくあげられますが，MINI-PASCALでは，32767までの整数しか使えず，階乗はすぐオーバーフローしてしまうため，1からNまでの自然数の和を求めるプログラムを再帰的呼び出しを用いて書いた例をあげてみましょう．もちろん，これはBASICでするようにFORを用いて書いてもかまいませんが，ここでは説明のためにあえてこの形であげておきます．

```
1   PROGRAM SAMPLE;
2   VAR N:INTEGER;
3   FUNCTION SUM(N:INTEGER);
4     BEGIN
5       IF N<1 THEN SUM:=0
6              ELSE IF N=1 THEN SUM:=1
7                          ELSE SUM:=SUM(N-1)+N
8     END;
9   BEGIN
10    WRITE('input N=? ');
11    READ(N);
12    WRITE('1+2+...+N = ',SUM(N))
13  END.
```

これは，自然数Nを入力し，1からNまでの自然数の和を求める関数SUM(N)のプログラムですが，中にSUM：=SUM(N-1)+Nという文があります．これは，関数SUMの値を求めるために，関数SUM自身の値を使っているわけですが，これが再帰的呼び出しなのです．これでは循環論法に陥り，いつまでたっても値が求められないのではないか，という疑問がわく方もあると思います．しかし，再帰的呼び出しをくり返すたびにNの値は減ってゆき，N＝1のときはSUM：＝1としてあるための無限ループには陥らないのです．ただ，この疑問は重要な点をついており，不用意なプログラムを書いたときには本当に無限ループに陥ってしまいますので注意して下さい．

では，このプログラムの働きについて，もう少し詳しくみてみましょう．Nに与えられた値を3としますと，SUM(3)の値を求めるためにFUNCTIONが呼び出されます，というように順を追

ってプログラムをみていきます.

① まず, FUNCTION　SUMが呼び出され, ローカル変数Nにはグローバル変数Nの値, 3が入ります.

② 次に, 関数実行部に入りますが, 今, N＝3ですから2つ目のELSEの後のSUM：＝SUM(N-1)＋Nを実行します.

③ そこで, N-1＝2なのでSUM(2)を計算するため, 自分自身であるFUNCTION　SUMを呼び出します. このとき, ローカル変数Nには今までのN-1の値すなわち2が入ります.

④ ②と同様, SUM：＝SUM(N-1)＋Nを実行します.

⑤ 今度はN-1＝1ですから, SUM(1)を計算するため, 再び自分自身を呼び出します. このとき, ローカル変数Nの値は今までのN-1の値, すなわち1となります.

⑥ 今度はN＝1ですから, 2つ目のTHENの後が実行され, SUM：＝1となり計算を終わって戻ります.

⑦ ④でSUM(N-1)の部分が1と計算されたところへ戻ってきます. ここでのNの値は2(1回FUCTIONから戻ってきたので回復されている)ですから, SUM：＝3ということになります. この値を持って再びFUNCTIONから戻ります.

⑧ 戻った先は, ②でSUM(N-1)の部分が3と計算されたところです. ここでのNの値は, 再び回復されて3になっていますので, SUM：＝6となってFUNCTIONから戻ります.

⑨ これで計算は全て終わったので, SUM(3)は6ということになります. 戻ってきたところでは, Nの値はグローバル変数の値3になっています.

多少複雑だったかもしれませんが, 上のプロセスで, Nが何回もローカル変数として値を与えられ, また, FUNCTIONから戻るたびに値が一段階ずつ回復してゆくところがポイントとなっていることがわかると思います.

変数の区別のないBASICでは, こういうことはできません. 無理に再帰的呼び出しをしようとすると, 1回自分自身を呼び出すごとに, 変数値を配列などにとっておかなくてはいけません.

こういったことが楽にできるのが, PASCALの強みです. 再帰的呼び出しは, 論理的なアルゴリズムにはたびたび現われるもので, PASCALのようにすっきりとわかりやすく書けるということは, 大変便利なことなのです. 例えば, 前にあげた文の構文図の中に, また文が現れるということがありましたが(図8-2-11), その場合, ある文字列が文であるかどうかを判別する手続きは, 再帰的呼び出しを使って書けばよいのです. 例えば文字列中のBEGINから"；"またはENDまでの文字列について, その手続き自身で文であるかどうかを判定するということです. 大変便利ですから, ぜひ利用法をマスターして下さい.

最後に, 10進数を4桁の16進数に変換するプログラム例をあげておきましょう. このようなプログラムはマニュアルにも出ていますが, ここでは, 必要に応じて0を補って4桁の16進数を文字列として与えるプログラムを, 再帰的呼び出しを使ってつくりました. また, 負の数も正しく

変換できるようになっています(8-2-5で説明しますが，MODにつごうの悪い性質があるため，負の数については別の処理を補うことが必要なのです)．では，再帰的呼び出しの効果をじっくり味わってみて下さい．前の例のように，働きを1段ずつ追ってみるのもいいかもしれません．

```
1     PROGRAM SAMPLE;
2     VAR I:INTEGER;
3         S:STRING;
4     PROCEDURE INTHEX1(I:INTEGER; VAR S:STRING);
5       VAR H,J:INTEGER;
6           T:STRING;
7       BEGIN
8         J:=I DIV 16;
9         H:=I MOD 16;
10        IF(I<0)AND(H>0)THEN H:=16-H;
11        IF(I<0)AND(H>0)THEN J:=J-1;
12        IF J=-1 THEN J:=0;
13        IF J<>0 THEN INTHEX1(J,S);
14        IF H>=10 THEN CHR(H+55,T)
15                 ELSE CHR(H+48,T);
16        INSERT(T,S,LENGTH(S))
17      END;
18    PROCEDURE INTHEX(I:INTEGER; VAR S:STRING);
19      VAR T:STRING;
20      BEGIN
21        S:='';
22        INTHEX1(I,S);
23        T:='000';
24        IF I<0 THEN T:='FFF';
25        INSERT(T,S,0);
26        DELETE(S,1,LENGTH(S)-4)
27      END;
28    BEGIN
29      WRITE('input value ?');
30      READ(I);
31      INTHEX(I,S);
32      WRITE('==> hex',/,'#',S,/)
33    END.
```

8-2-4　エラーメッセージについて

MINI-PASCALでは，プログラム実行時のエラーは，すべてエラー番号で表示されるため，少しわかりにくくなっています．そこで，エラーメッセージが出た場合の対応等について少し説明しましょう．まず，RUNをかけたとき，プログラムを実行する前にエラーチェックを行うため，エラーがあればたいていの場合何もプログラムを実行しないうちに，エラーメッセージが出ます．プログラム実行中のエラーは，0で割った(エラー番号25)ときや，配列で範囲外を指定した(エラー番号24)ときや，変数型が入力の際一致していない(エラー番号26)ときなどでわかりやすいのですが，めんどうなのは〝;〟にかかわるエラーです．特に，BASICに慣れている人の場合〝;〟を打たずにRETキーを押してしまうことがあるため，エラー番号5，10，19等がしばしば現れてしまいます．また〝;〟を忘れると，8-1-3でも説明したように，次の文とつながっていると解釈されてしまいますので，解釈がどんどんずれて変なところで〝;〟がない，というエラーメッセージの出ることがあります．そんなとき，エラーメッセージとともに出てきた行だけを見ていてもわかりませんので，全体にわたって〝;〟が正しい位置にあるかどうかをよく確かめて下さい．また，未定義の変数名をFORの後に使ったときは，名前が未定義(エラー番号11)ではなく，

FOR文の誤り(エラー番号18)として出るので注意して下さい．変数名の宣言にも注意が必要です．

8-2-5　その他の注意点

MINI-PASCALを使う際に注意しなければならない点，まちがいやすい点等について，いくつかあげておきますので，プログラム開発の際に役立てて下さい．

① 配列について

配列は，整数型しかとれませんが，添字範囲は1からです．BASICと違い0番めの要素というものはなく，指定するとエラー番号24が出てしまいます．

② 整数型変数のREADについて

READ文で変数値を読み込むとき，整数型を要求しているのにRETキーを押すと，エラー番号26が出てしまいます．これも，BASICと違いますので注意して下さい．数値の前には，+，－の他に#も付けることができ，16進数の入力が可能です．入力した文字列に，数字0～9以外のものが出てくると区切りとみなし，その前までを有効な入力とします．したがって，1つのREAD文で複数の変数値を要求しているときは，変数値の区切りには0～9以外のすべての文字が使えます．1つのREAD文で要求している変数の数より，入力した値の数が少ないときは足りない変数に値0が入り，入力した値の数が多いときは余った値は無視されます．例えば，READ(A，B)という文に(A，Bは整数型とします)#100#100#100と入力すると，Aには256，Bには100が値として入ります．はじめの#は16進数の記号，後の2つの#は区切り記号とみなされ，最後の100は無視され，2番めの100がBの値になるのです．

③ ファイングラフィックモードについて

SCREEN(2)でファイングラフィックモードとなるわけですが，T-BASICと違いこのモードでもカラーが使えます．文法説明でPSETやLINEに，実際にカラーコードが付いていることが確かめられるでしょう．ただ，パソピアのファイングラフィックモードは，色を付ける部分の左側1キャラクタ分にカラーコードをおく，という方式なので横方向をつめて違う色で線を書きたいという場合にうまくいきません．簡単に色が付けられるようになったのはよいのですが，思い通りの画面にならず困ってしまうということもありますので気を付けて下さい．

④ 整数型変数の値について

整数型変数の値は，2バイトの2の補数表示で表されており，0から1つずつ数字を増していくと，

0→1→……→32767→－32768→－32767→……→－1→0
→…

というように変化していきます．(初期のマニュアルの説明は誤っています)．この際，オーバ

ーフロー表示などはなく，繰り上がり，繰り下がりなどは無視して計算を行います．2進法表示のよくわからない方は，32768（$=2^{15}$）以上の数字は65536の倍数を引いた数字として扱われると考えて下さい．つまり，65535は65535－65536＝－1より－1として扱われます．したがって，256＊256＝0，256＊128＝－32768となり，READ（A）に対し（Aは整数型とします）65535と入力すると，Aの値は－1となってしまいます．例えば，メモリのサーチなどをするつもりでFOR　I：＝0　TO　#FFFF　DO～などと書くと，0から－1までと解釈されるのでDO以下は1回も実行されなくなってしまいます．（BASICではこのようなとき，1回実行してしまうものもありますが，MINI-PASCALでは実行しません）．このように注意しないと予想外の結果を招いてしまいますので，32768以上の数（負の数）を扱うときは気を付けて下さい．

宣言しただけで，何も代入されていない整数型変数の値には0が入っています．しかし，はじめに0を代入したいときでもきちんと0を代入する文を書くようにしておいた方が誤りが少ないでしょう．

⑤　MOD，DIVについて

MODは余りを，DIVは整数の商を表しますが，ここでも負の数のかかわるものはめんどうです．例えば，5　MOD　3＝2，5　DIV　3＝1ですが，次の値はいくつになると思いますか？

①　（－5）MOD　3
②　5　MOD（－3）
③　（－5）MOD（－3）
④　（－5）DIV　3
⑤　5　DIV（－3）
⑥　（－5）DIV（－3）

正解は①から順番に，2，2，2，－1，－1，1となります．MODでは，マイナス記号はあってもなくても関係ないということです．（このため，P8-2-8ではめんどうな処理をしました）①に対して期待される答は，－2か1ですので，ここで出た答は少し不適当といえるでしょう．また，DIVでは整数として割り切れない答を整数にするとき，0に向って切り捨てるということに注意して下さい．例えば－5÷3＝－1.66…ですが，四捨五入でもなく，またBASICのINTのように小さい方へ切り捨てるのでもなく，0に近い方へ切り捨てて－1となります．

⑥　LOAD，SAVE

MINI-PASCALの初期のバージョンには，カセットによるLOAD/SAVEがうまくできないというバグがありました．しかし，現在，販売されているMINI-PASCALには，そのような不備な点は改良されています．初期のバージョンをお持ちの方は度々迷惑をこうむったことと思います．東芝ではそのような方々のために，無償で現在のバージョンと交換する便宜をはかっていますの

で，利用されることをお勧めします．

8-2-6　MINI-PASCAL内部の状態

ここでは，MINI-PASCALのプログラムの格納のされ方，内部サブルーチンについて説明します．どちらも，詳しくは説明できませんが，参考程度にお話ししておきます．

プログラムの格納のされ方ですが，プログラムは内部で中間コードとして格納されています．その方式は，大きく分けて次の表のように5通りあります．

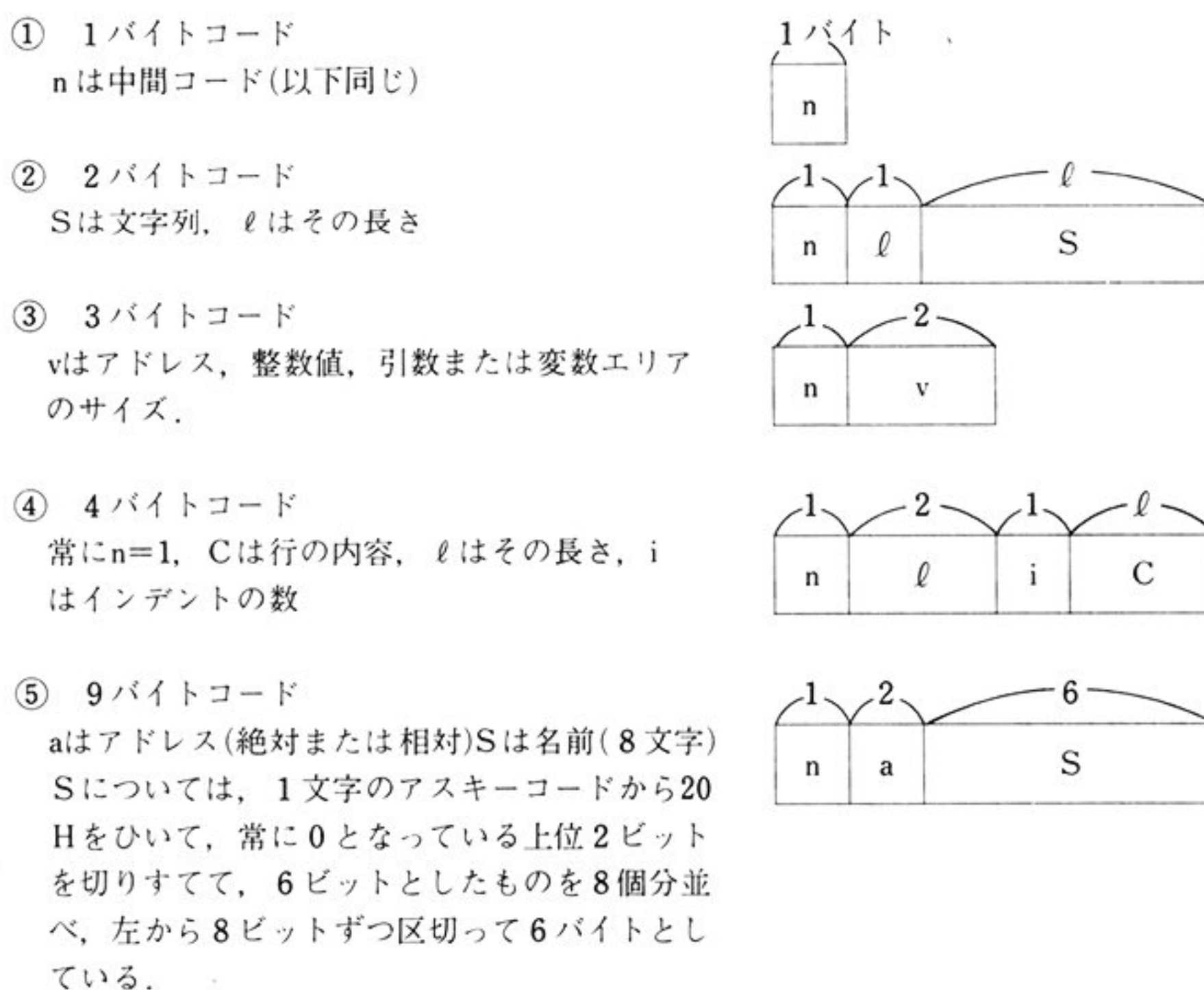
図表8－2－7　MINI-PASCAL中間コードの形式

① 1バイトコード
nは中間コード(以下同じ)

② 2バイトコード
Sは文字列，ℓはその長さ

③ 3バイトコード
vはアドレス，整数値，引数または変数エリアのサイズ．

④ 4バイトコード
常にn=1，Cは行の内容，ℓはその長さ，iはインデントの数

⑤ 9バイトコード
aはアドレス(絶対または相対)Sは名前(8文字)Sについては，1文字のアスキーコードから20Hをひいて，常に0となっている上位2ビットを切りすてて，6ビットとしたものを8個分並べ，左から8ビットずつ区切って6バイトとしている．

予約語，特殊記号の中間コードを，次の図表にあげておきます．コードは10進数で表し，形式は図表8-2-1における①～⑤のいずれの形式をとるかを示しています．また，図表8-2-1のS，v，Sが何を表すかを，②，③，⑤の形式についてあげておきます．

図表8－2－8　MINI PASCAL中間コード表

コード	形　式	意　　味	
0	①	プログラム終了	
1	④	行開始	
18	①	BEGIN	
19	③	DO	ループ脱出後の区切りのアドレス
20	①	DOWNTO	

コード	形 式	意 味	
21	③	ELSE	IF文脱出後の区切りのアドレス
22	①	END	
23	①	FOR	
24	③	FUNCTION	仮引数エリアサイズ
25	①	IF	
26	①	INTEGER	
27	③	PROCEDURE	仮引数エリアサイズ
28	③	PROGRAM	ボディー部の先頭アドレス
29	①	REPEAT	
30	①	STRING	
31	③	THEN	ELSE部の先頭アドレス
32	①	TO	
33	①	UNTIL	
34	③	VAR	変数エリアサイズ
35	①	WHILE	
36	①	(	
37	①	)	
38	①	+	
39	①	−	
40	①	OR	
41	①	XOR	
42	①	*	
43	①	AND	
44	①	DIV	
45	①	MOD	
46	①	NOT	
47	①	〔	
48	①	〕	
49	①	,	
50	①	;	
51	①	.	
52	①	=	
53	①	<	
54	①	>	
55	①	<=	
56	①	>=	
57	①	< >	
58	②	コメント	文字列
59	②	文字列定数	文字列
60	③	整定数	整数値
61	③	16進定数	整数値
62	①	/	
63	①	:=	
64	①	:	
199	⑤	識別子	未定

コード	形 式	意 味	
200	⑤	標準手続名	手続の先頭アドレス
201	⑤	手続名	〃
202	⑤	標準関数名	関数の先頭アドレス
203	⑤	関数名	〃
204	⑤	グローバル整変数	先頭アドレス
205	⑤	グローバル整変数配列	〃
206	⑤	グローバル文字変数	〃
208	⑤	ローカル整変数	〃
209	⑤	ローカル整変数配列	〃
210	⑤	ローカル文字変数	〃

これらの格納状態を示す例として，プログラムをあげておきます．

```
1     PROGRAM MONITOR;
2     VAR I,J,K:INTEGER;
3     BEGIN
4       I:=32367;
5       REPEAT
6         FOR J:=I TO I+15 DO
7           BEGIN
8           K:=PEEK(J);
9           IF K>99 THEN WRITE(' ',K)
10                  ELSE IF K>9 THEN WRITE('  ',K)
11                               ELSE WRITE('   ',K)
12          END;
13        WRITE(/);
14        I:=I+16
15      UNTIL I>32500
16    END.
```

MINI-PASCALでは，命令を直接実行することができないため，内部の格納状態を見る場合，どうしてもそのプログラムをMINI-PASCALで書かなくてはなりません．そのため，そのプログラムで読むことができるのは，そのプログラム自身の格納状態だけということになります．

内部のメモリ状態をダンプする機械語サブルーチンをつくっても，それをコールするプログラムの格納状態しか見られないことにかわりありません．ですから，すべての格納状態をこのプログラムで知るのは難しいことですが，前にあげた表とあわせれば，だいたいのことはわかると思います．つまり，このプログラムをRUNさせると，このプログラム自身の内部の格納状態が10進数で，途中までですが順々に出てきます．はじめのところだけ少し説明すると，28はPROGRAMの中間コード，次の2バイト(167, 126)はアドレス，199は識別子の中間コード，次の2バイト(0, 0)は未定のアドレス，その次から6バイトは"MONITOR"の6ビットコード，2D, 2F, 2E, 29, 34, 2F, 32, 00をつなげて8ビットずつ区切ったものになります．

APPENDIXに，MINI-PASCALの内部ルーチンの一覧表を載せましたので，興味のある方は参考にして下さい．

8-3　T-BASICとの比較

8-3-1　各命令について

今までも，MINI-PASCALの説明をするにあたって，T-BASICと比較して述べたところがいくつかありますが，ここでは，もっと詳しくT-BASICと比べてみたいと思います．まずはじめに，T-BASICとMINI-PASCALでのプログラムの書き方の差についてまとめておきましょう．(表1，表2参照)

表1　T-BASICとMINI-PASCALの比較

形　式	T-BASIC	MINI-PASCAL
①文の区切り	"：" を使用 改行したところでは不要	"；" を使用 改行したところでも必要
②使用する変数等	特に宣言する必要はない． サブルーチン，DEF　FNは，どこにおいてもよい．	使用する変数は，前もって名前，型を宣言する必要がある． PROCEDURE，FUNCTIONは，はじめに宣言する必要がある．
③文字列をはさむもの	" を使用	' を使用
④文の頭部	特にきまっていない	PROGRAM文が必要
⑤コマンドの書き方	コマンドの後，すぐオペランドがくることが多い．	オペランドは，常に "("と")" でくくる必要がある
内容的な違い		
①GOTO文	あり	なし
②ローカル変数とグローバル変数	なし	あり
③エラーチェック	エラーのある部分を実行したときにはじめて，エラーメッセージが出る．	実行前にひととおり，エラーチェックを行う．
④直接命令	行番号を付けずに直接命令が実行可能.	直接実行はできない．

表2　T-BASICとMINI-PASCALの比較(命令)

T-BASIC	MINI-PASCAL
FOR～NEXT	FOR～TO(DOWNTO)～DO～
IF～THEN～ELSE～	IF～THEN～ELSE～
WHILE～WEND	WHILE～DO～
なし	REPEAT～UNTIL～

次に，MINI-PASCALの標準手続き，関数とT-BASICでそれに相当するものとの対応をまとめておきます(表3参照).

MINI-PASCAL	T-BASIC
CHR (I, S)	S$=CHR$ (I)
INSERT (S1, S2, I)	S2$=LEFT$ (S2$, I)+S1$+RIGHT$ (S2$, LEN(S2$)−I)
DELETE(S, I, J)	S$=LEFT$(S$, I−1)+RIGHT$(S$, LEN(S$)+1−I−J)
READ(〜)	INPUT 〜
WRITE(〜)	PRINT 〜
GOTOXY(X, Y)	LOCATE X, Y
GETKEY(C)	C$=INKEY$
POKE(A, D)	POKE A, D
OUT(P, D)	OUT P, D
MOVE(S1, I, S2, J, K)	S2$=LEFT$(S2$, J−1)+MID$(S1$, I, K)+RIGHT$(S2$, (LEN(S2$)+1−J−K)*(1+(LEN(S2$)+1−J−K<0)))
CALL(A; P1, P2, ……, Pn)	DEF USR=A : A=USR(P1) (複数のデータは直接わたせません)
COLOR(I, J)	COLOR I, J
LINE(X1, Y1, X2, Y2, 'PSET', 5, 'B')	LINE(X1, Y1)−(X2, Y2), 5, B (ファイングラフィックモードの場合，色指定ができません)
PSET(X, Y, C)	PSET(X, Y), C (LINEと同じ注意があてはまります)
PRESET(X, Y)	PRESET(X, Y)
SCREEN(M)	SCREEN M
WIDTH(C, L)	WIDTH C (行数は変わりません，そのかわりCとして，1〜80の任意の値を指定できます)
ORD(S)	ASC(S$)
LENGTH(S)	LEN(S$)
PEEK(A)	PEEK(A)
INP(P)	INP(P)
ADR(A)	VARPTR(A)
POINT(X, Y)	POINT(X, Y)
TIME	TIME

表3 MINI-PASCALとT-BASICの関数，コマンド対応表

以上の表を標準手続き，関数については，文字列操作以外はT-BASICとそんなに違いません．逆にT-BASICの方からMINI-PASCALを見ると，整数型のため当然三角関数や指数対数関数がなくその他おもなものでは，RND, GOTO, INSTR, HEX$, READ, DATA, STR$, VALなどに対応するものがなくなっています．

8-3-2　ベンチマークテスト

ここでは，MINI-PASCALとT-BASICの能力比較ということで，ベンチマークテストをしてみました．ベンチマークテストとは，ある決まったプログラムを実行させてみて，その所要時間をはかるものです．MINI-PASCALは，PASCALの教育用システムと考えるべきものですから，別にスピードにこだわる必要もありませんが，参考までに御覧下さい．使用したプログラムは，T-BASICではリスト1～11，MINI-PASCALでは，リスト12～22です．

〈リスト1〉

```
300 PRINT "START"
400 FOR K=1 TO 1000
500 NEXT K
700 PRINT "END"
800 END
```

〈リスト2〉

```
300 PRINT "START"
400 K=0
500 K=K+1
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
```

〈リスト3〉

```
300 PRINT "START"
400 K=0
500 K=K+1
510 A=K/K*K+K-K
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
```

〈リスト4〉

```
300 PRINT "START"
400 K=0
500 K=K+1
510 A=K/2*3+4-5
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
```

〈リスト12〉

```
1     PROGRAM TEST1;
2     VAR K:INTEGER;
3     BEGIN
4       WRITE('START',/);
5       FOR K:=1 TO 1000 DO;
6       WRITE('END',/)
7     END.
```

〈リスト13〉

```
1     PROGRAM TEST2;
2     VAR K:INTEGER;
3     BEGIN
4       WRITE('START',/);
5       K:=0;
6       REPEAT
7         K:=K+1
8       UNTIL K>=1000;
9       WRITE('END',/)
10    END.
```

〈リスト14〉

```
1     PROGRAM TEST3;
2     VAR A,K:INTEGER;
3     BEGIN
4       WRITE('START',/);
5       K:=0;
6       REPEAT
7         K:=K+1;
8         A:=K DIV K*K+K-K
9       UNTIL K>=1000;
10      WRITE('END',/)
11    END.
```

〈リスト15〉

```
1     PROGRAM TEST4;
2     VAR A,K:INTEGER;
3     BEGIN
4       WRITE('START',/);
5       K:=0;
6       REPEAT
7         K:=K+1;
8         A:=K DIV 2*3+4-5
9       UNTIL K>=1000;
10      WRITE('END',/)
11    END.
```

〈リスト5〉

```
300 PRINT "START"
400 K=0
500 K=K+1
510 A=K/2*3+4-5
520 GOSUB 820
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

〈リスト6〉

```
300 PRINT "START"
400 K=0
430 DIM M(5)
500 K=K+1
510 A=K/2*3+4-5
520 GOSUB 820
530 FOR L=1 TO 5
540 NEXT L
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

〈リスト7〉

```
300 PRINT "START"
400 K=0
430 DIM M(5)
500 K=K+1
510 A=K/2*3+4-5
520 GOSUB 820
530 FOR L=1 TO 5
535 M(L)=A
540 NEXT L
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

〈リスト8〉

```
100 SCREEN 1
200 CLS
300 PRINT "START"
400 FOR I=1 TO 1000
500 PSET (1,1),1
600 PRESET (1,1)
700 NEXT I
800 PRINT "END"
900 END
```

〈リスト16〉

```
1     PROGRAM TEST5;
2     VAR A,K:INTEGER;
3     PROCEDURE GOSUB;
4     BEGIN
5     END;
6     BEGIN
7       WRITE('START',/);
8       K:=0;
9       REPEAT
10        K:=K+1;
11        A:=K DIV 2*3+4-5;
12        GOSUB
13      UNTIL K>=1000;
14      WRITE('END',/)
15    END.
```

〈リスト17〉

```
1     PROGRAM TEST6;
2     VAR A,K,L:INTEGER;
3         M:INTEGER[5];
4     PROCEDURE GOSUB;
5     BEGIN
6     END;
7     BEGIN
8       WRITE('START',/);
9       K:=0;
10      REPEAT
11        K:=K+1;
12        A:=K DIV 2*3+4-5;
13        GOSUB;
14        FOR L:=1 TO 5 DO
15      UNTIL K>=1000;
16      WRITE('END',/)
17    END.
```

〈リスト18〉

```
1     PROGRAM TEST7;
2     VAR A,K,L:INTEGER;
3         M:INTEGER[5];
4     PROCEDURE GOSUB;
5     BEGIN
6     END;
7     BEGIN
8       WRITE('START',/);
9       K:=0;
10      REPEAT
11        K:=K+1;
12        A:=K DIV 2*3+4-5;
13        GOSUB;
14        FOR L:=1 TO 5 DO M[L]:=A
15      UNTIL K>=1000;
16      WRITE('END',/)
17    END.
```

〈リスト19〉

```
1     PROGRAM TEST8;
2     VAR I:INTEGER;
3     BEGIN
4       SCREEN(1);
5       WRITE('START',/);
6       FOR I:=1 TO 1000 DO
7         BEGIN
8           PSET(1,1,1);
9           PRESET(1,1)
10        END;
11      WRITE('END')
12    END.
```

〈リスト9〉
```
100 SCREEN 2
200 CLS
300 PRINT "START"
400 FOR I=1 TO 1000
500 PSET (1,1),1
600 PRESET (1,1)
700 NEXT I
800 PRINT "END"
900 END
```

〈リスト20〉
```
1     PROGRAM TEST9;
2     VAR I:INTEGER;
3     BEGIN
4       SCREEN(2);
5       WRITE('START',/);
6       FOR I:=1 TO 1000 DO
7         BEGIN
8           PSET(1,1,1);
9           PRESET(1,1)
10        END;
11      WRITE('END')
12    END.
```

〈リスト10〉
```
100 SCREEN 1
200 CLS
300 PRINT "START"
400 FOR I=1 TO 50
500 LINE (0,0)-(159,0),1
600 NEXT I
700 PRINT "END"
800 END
```

〈リスト21〉
```
1     PROGRAM TEST10;
2     VAR I:INTEGER;
3     BEGIN
4       SCREEN(1);
5       WRITE('START',/);
6       FOR I:=1 TO 50 DO
7         LINE(0,0,159,0,'pset',1,'');
8       WRITE('END')
9     END.
```

〈リスト11〉
```
100 SCREEN 2
200 CLS
300 PRINT "START"
400 FOR I=1 TO 50
500 LINE (0,0)-(639,0),1
600 NEXT I
700 PRINT "END"
800 END
```

〈リスト22〉
```
1     PROGRAM TEST11;
2     VAR I:INTEGER;
3     BEGIN
4       SCREEN(2);
5       WRITE('START',/);
6       FOR I:=1 TO 50 DO
7         LINE(0,0,639,0,'pset',1,'');
8       WRITE('END')
9     END.
```

このうち1番目から7番目までは,月刊アスキーでいつも使っているベンチマークテスト用プログラムで,8番目から11番目までは,グラフィックの能力を測るため加えたものです.そして,MINI-PASCALのプログラムは,それをできるだけ忠実に翻訳したものです.本来は,1番目から7番目の次に小数演算を含む8番目のテスト用プログラムがあるのですが,MINI-PASCALでは,小数が扱えないためカットしてあります.これらのプログラムを実行した際の所要時間は,図8-3-1にまとめてあります.

見ての通りMINI-PASCALとT-BASICは,ほぼ互角です.もともとT-BASICは他機械種に比べ十分速いので,MINI-PASCALは,スピードに関しては合格点といえるでしょう(現在,T-BASICはおもに使われている他機種に比べ,ほぼ1.5倍のスピードです).

グラフィック関係のプログラムに関しては,T-BASICにはCLSが入っていますが,MINI-PASCALにはCLSに相当するものがなく,SCREENを実行したときに画面がクリアされるため,入っていません.また,T-BASICでは,SCREEN文を実行したときに,モード変更があると画面をクリアしてしまい時間がかかるため,グラフィックないしはファイングラフィックモードにしてから実行してあります.画面の一行あたりの文字数は,どちらも80字でテストしました.

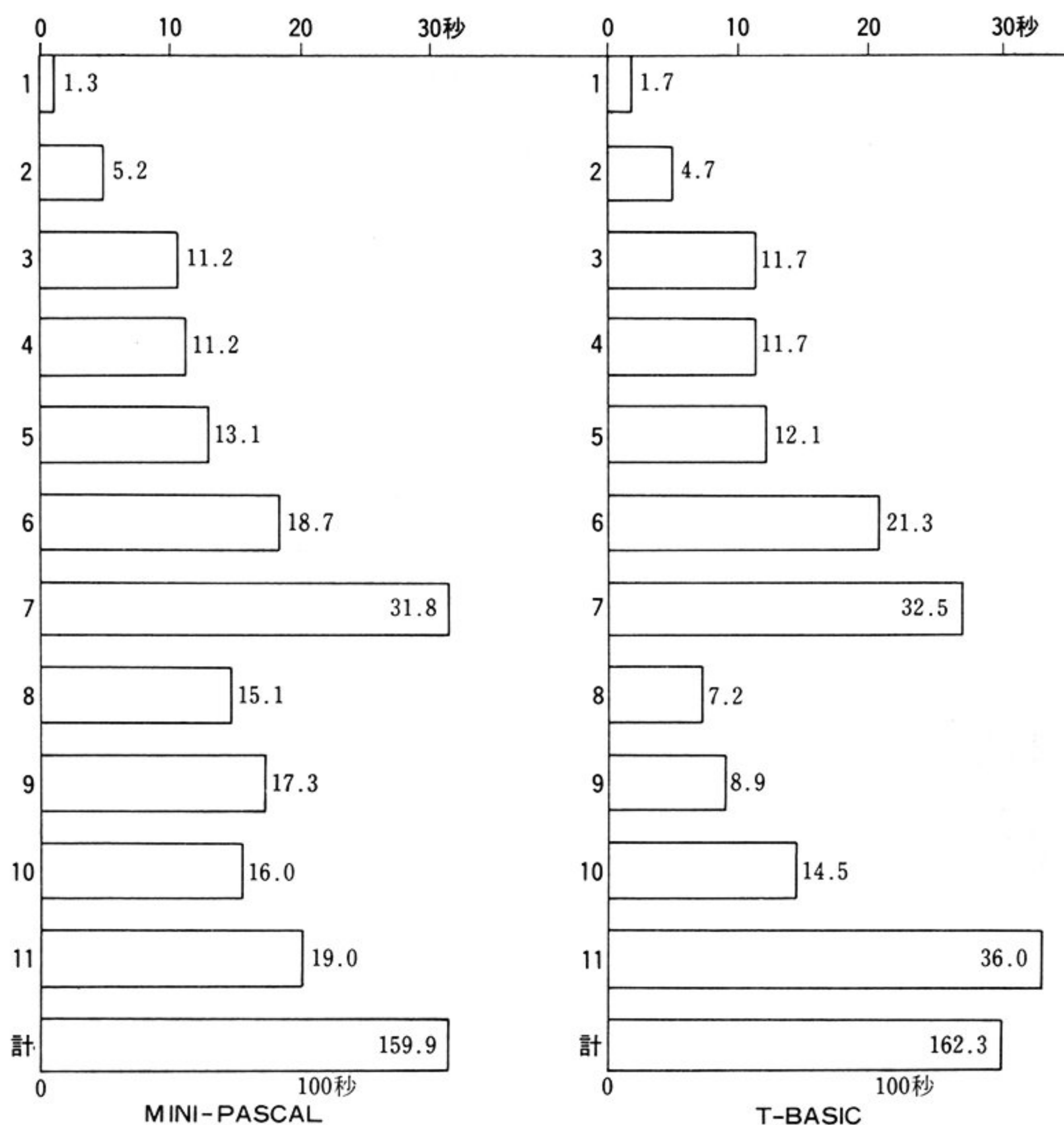



図8－3－1　ベンチマークテスト

8-3-3　T-BASICとMINI-PASCALの総合的比較

最後に，T-BASICとMINI-PASCALの総合的な比較をしておきましょう．スピードに関しては，どちらもあまり差はありませんでした．プログラムの読み易さは，MINI-PASCALが勝っています．しかし，使える変数型の豊富さ，エディットのしやすさ等は，T-BASICが勝っています．MINI-PASCALは，エラーメッセージがわかりにくく，BASICでは簡単にダイレクトモードで実行できるようなことも，いちいちPROGRAMやVAR, BEGIN～ENDなどと書かなくてはなりません．したがって，手軽にプログラムをつくるにはT-BASICの方が使い易く，論理的にすっきりと見やすいプログラムをつくるにはMINI-PASCALの方が適している，と言えるでしょう．

このように，MINI-PASCALとT-BASICのどちらが優れている，というようなことは一口には言えませんが，次代の中心となるであろうPASCALが，安価な教育システムとしてPASOPIAで使えるようになった，ということは喜ぶべきことと言えます．

第9章

CP/M

第9章　CP／M

9-1　CP／Mの概略

9-1-1　CP／Mとは？

この章では，パソピア-CP/Mに関する説明を行うことにしましょう．

CP/Mという言葉を聞いたことがない，聞いたことはあるが何のことかよく知らない，という人も少なくないと思います．CP/Mというのは，Digital-Research社によって開発された，ディスクを管理するシステムプログラム（DOS-Disk Operating System）の名前であり，Control Program for Microcomputersの頭文字からつけられたものです．このプログラムは，さまざまな機種に合わせて発売されています．そのうち東芝パソピア用にインプリメントされたものが，ここで説明するパソピア-CP/Mです．

このように，多くの機種で採用されているプログラムとして，すぐ思いつくのはBASICのインタプリタですが，CP/Mは，それとは性質が違います．BASICは，各機種ごとに別なインタプリタがつくられているため，機械によって使える命令が違ったり，文法が違ったりすることがしばしばあります．同じBASICという名前がついても，全く別のプログラミング言語のように見えるものさえあるのです．そのため，プログラミング解説書，ソフトウェアパッケージなどは，機種別に製作・販売されているわけで，他機種のものを誤って買ってしまうと，あまり（あるいは全く）役に立ちません．

ところが，CP/Mは違います．CP/Mは，命令を解釈・実行したりする本体は，どの機械のものも同一のプログラムであり，入出力など機械ごとに変更しなくてはならない部分だけを，差し替えるようになっているのです．（その他，機種によっては拡張されているものもあります）．そのため使い方は，どの機種でも同一と考えてよく，プログラムも，CP/M上で走るものなら異なる機種間でも通用するわけです．ですからCP/Mは，標準的なDOSとして，世界中で広く使われているのです．あなたのパソピアも，CP/Mによって大きく世界が広がることでしょう．

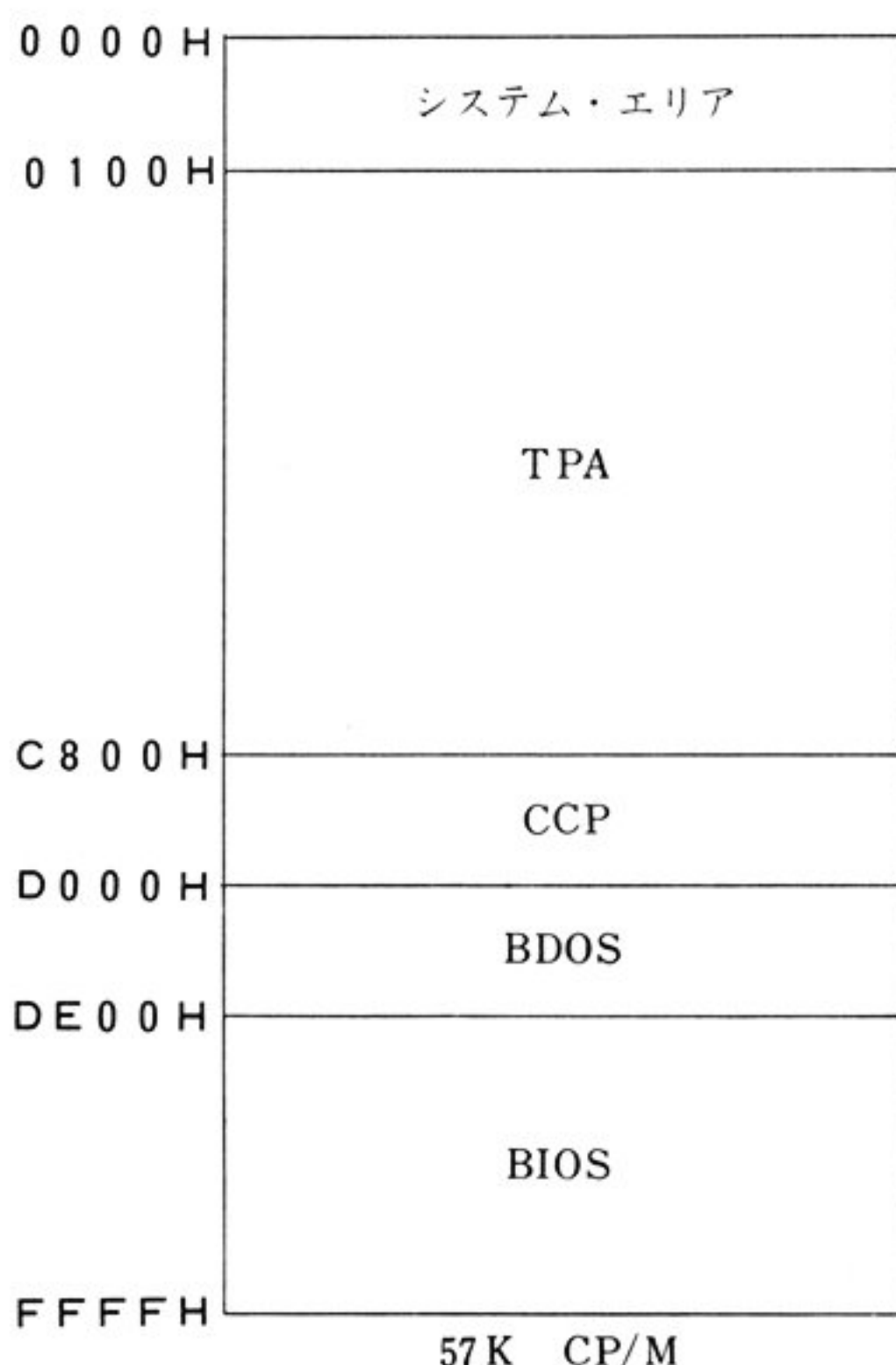
なお，上に述べたように，CP/Mの本体のプログラムは，どの機械でも同じなので，そのCPUも同じ機械語が走るものでなくてはなりません．CP/Mは，もともと8080向けに開発されたものですが，パソピアのCPU，Z80は，8080の機械語プログラムがそのまま動くので，その点，問題はあ

りません．しかし，現在日本で手に入るパーソナル・コンピュータにも，8080，Z80以外に6809，6502などのCPUを使っているものが少なくありません．このような機械では，そのままではCP/Mが走らせられないため，わざわざZ80のプログラムを走らせるZ80カードを，オプションで販売しています．Z80のプログラムは数多いので，何もCP/Mだけのために売られているわけではありませんが，主な目的は，やはりCP/Mといってよいでしょう．

CP/Mとは，わざわざそれだけのことをしてまで，走らせたいほどの価値あるプログラムなのです．

なお，この章では，紙面のつごうでCP/Mのすべてを解説することはできませんでした．もっと詳しく知りたいという方のために㈱アスキーより「入門CP/M」，「実習CP/M」，「応用CP/M」のCP/M 3部作が出版されているので参考にして下さい．

9-1-2　メモリ・マップ

ここでは，CP/Mが走っているときのメモリの内容のあらましを説明しましょう．パソピアのT-BASICでは，0～7FFFH番地までは，ROMになっていて，BASICのインタプリタが入っているのですが，CP/Mは，常にプログラムをディスクとやりとりしなければならないため，どの機械でもオールRAMにして使うことになっています．ですから，パソピアでもバンク切換を行って，0～7FFFH番地もRAMにしてしまうわけです．そのときのメモリ・マップを次に掲げましょう．

図表9－1－1　CP/M メモリ・マップ

この図を説明しましょう．はじめのシステム・エリアというのは，CP/Mが使用するメモリであり，ジャンプ・テーブル，ワークエリアなどになっています．このエリアは，ユーザーが勝手に使ってはいけないところです．その次のTPAというのは，Transient Program Areaの略であり，ディスク上のプログラム，データなどは，このエリアにロードされて実行されます．このようにRAM上に，そのたびごとにプログラムをディスクからロードするため，多彩な応用が可能になるわけです．

その次のCCPとは，Console Command Processorの略で，キーボードから入力された簡単な命令を解釈・実行するプログラムが入っています．ごく基本的な部分まで，毎回TPAにロードしているのでは大変なので，ここに固定されたプログラムエリアがあるのです．TPAにロードされるプログラムと，CCPに固定されているプログラムの違いについては，9-1-3　CP/Mコマンドの実行のされ方，9-3-1　ビルトインコマンド，9-3-2　トランジェントコマンドを参照して下さい．

さて，また図表9-1-1にもどって，次のBDOSに移りましょう．これは，Basic Disk Operating Systemの略で，ディスクのファイル処理などを行うプログラムが入っています．そしてその下のBIOSは，Basic I/O Systemの略であり，CP/Mの入出力を扱うプログラムが入っています．CP/Mをある機種にインプリメントするには，この部分をそのハードウェアに合わせて書き替えればよいのです．したがってここには，パソピア用に開発された入出力プログラムが入っており，周辺機器とのやりとりを行っているわけです．

9-1-3　CP／Mコマンドの実行のされ方

前の節で説明したように，CP/MコマンドはCCPで直接解釈・実行されるものと，ディスクからプログラムをTPAにロードしてから，解釈・実行されるものの2種類があります．

前者をビルトインコマンド，後者をトランジェントコマンドと呼びます．これらについては，9-2-1 ビルトインコマンド，9-2-2 トランジェントコマンド，で説明しますので，ここでは，どのように命令が実行されるかという全体的な流れを説明しましょう．

まず，DIRという命令を取り上げます．これは，ビルトインコマンドの1つであり，ディスク上のファイル名を出力するものです．(DISK　BASICのFILESに相当)そこで，キーボードからDIRと打ち込んで，RETURNキーを押すと，これをCCPが解釈し，ディスク上のファイル名を出力して来る，というわけです．（写真9-1-1）

これに対し，ビルトインコマンドでないものを打ち込むと，CCPはそれをトランジェントコマンドと解釈し，その命令を解釈・実行するプログラム(ディスク上にそのコマンドと同じ名前でセーブされています)をロードしてきて，そのプログラムに制御を移します．

ここで，でたらめにＡＢＣなどと打ち込んでみましょう．ＡＢＣ？　と表示がでるはずです．これは，ＡＢＣをトランジェントコマンドとして解釈し，その名前のファイルをロードしようとしたのですが，そのようなファイルが存在しなかったために，ＡＢＣ？　という表示を出してきたわけです．そこで今度は，実際にある命令をためしてみます．COLOR 2 と打ち込んでみましょ

う．文字の色が赤色になります．

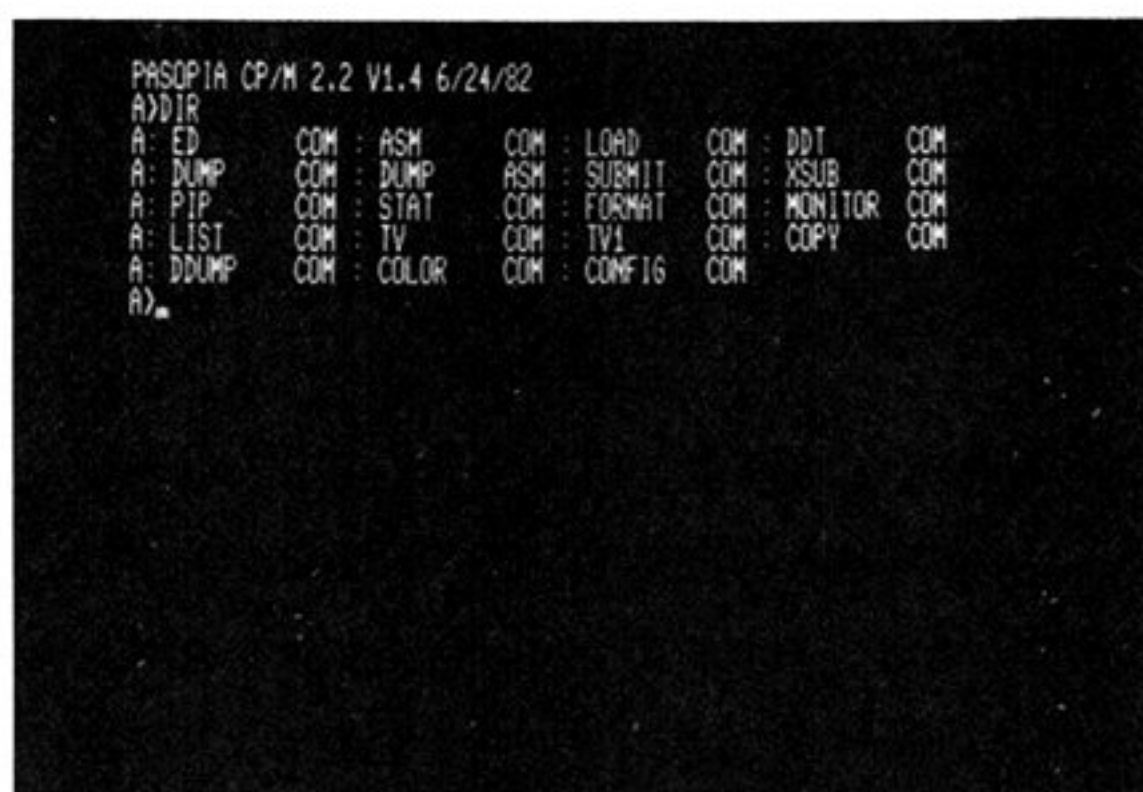



写真9－1－1
DIRの実行

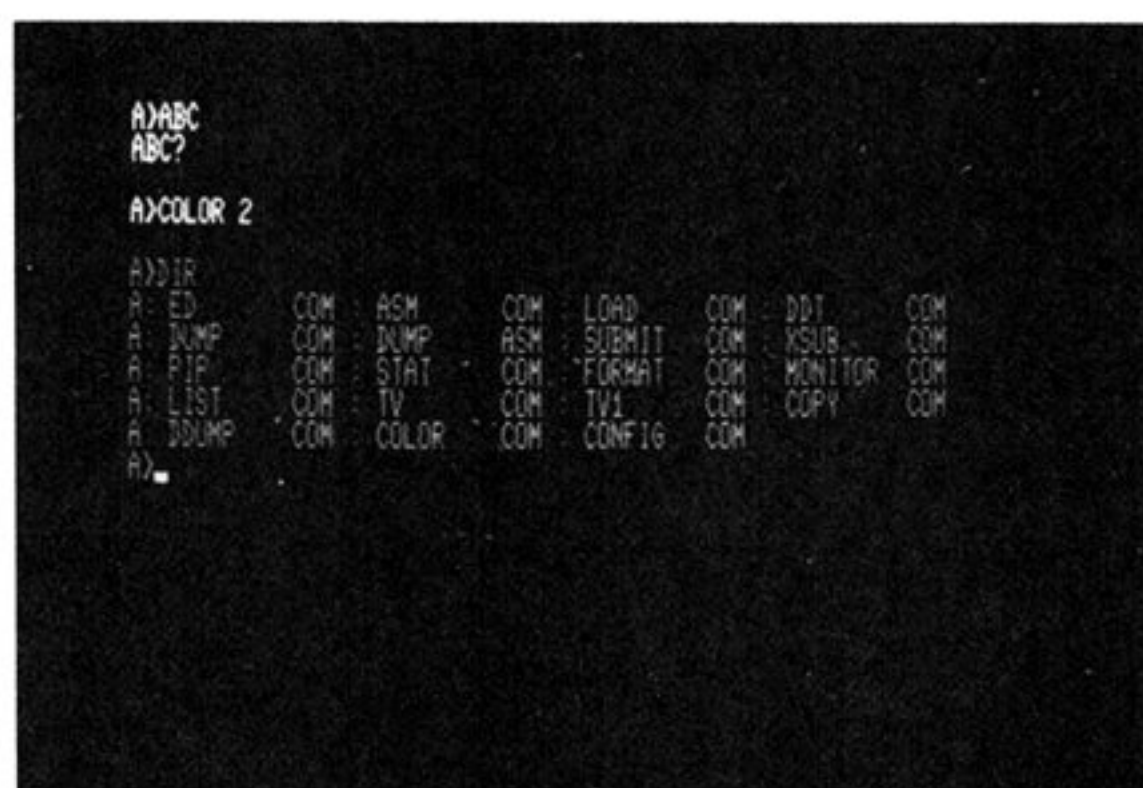



写真9－1－2
ABC？　COLOR2の実行

写真9－1－2は，CCPがCOLORという命令をトランジェントコマンドとして解釈し，その名前のファイルをディスクからロードしたものです．その結果プログラムでパラメータ2を解釈して，色を赤色に変えるという実行をしたのです．COLORというプログラムのファイルは，DIRを実行してみれば，COLOR.COMという形で見つかります．(写真9－1－1を見て下さい)．このCOMというのは，コマンドであることを表すものです．

さて，このようにトランジェントコマンドが実行されるわけですが，このCOLORというのは，CP/Mに標準的に組み込まれているコマンドではありません．これは，CP/Mの解説書を見ればわかりますが，カラーの出ないマイクロ・コンピュータもあるのですから当然のことです．今，実行できたのは，ディスク上にCOLOR.COMというファイルがあったからで，ディスク上に存在するこのようなファイルが，トランジェントコマンドなのです．ですから，自分で新しい命令を追加するには，コマンドとして実行可能なプログラムファイルを作って，ディスク上にセーブしておけばよいのです．このようなファイルの作り方は，9－2－2トランジェントコマンドのLOADで説明します．この機能は，いったん作ってしまえばもとからあるコマンドと全く同じように使えて，たいへん便利なものといえるでしょう．

9-2　システムに含まれる標準コマンド

9-2-1　ビルトインコマンド

ここでは，9-1-3で説明したビルトインコマンドというものについて，具体的に説明していきます．まず，ビルトインコマンドというのは，数は多くなくERA，DIR，REN，SAVE，TYPEの5つだけです(他にUSRがありますが，ここではふれません)．これらは，いずれもディスク上のファイルに関する命令で，順番にそれぞれファイルの抹消，ファイル名の出力，ファイル名の変更，TPAからディスクへのセーブ，ファイルの中身の出力を意味します．これらの5つの命令では，いずれもファイル名を指定する必要がありますが，CP/Mでは，ファイル名の指定のしかたには2種類の方法があります．これらは，トランジェントコマンドの一部でも用いられるので，簡単に説明しておきましょう．

まず，ファイル名を指定する際には，そのファイルの入っているディスクのドライブ名：ファイル名，という形をとります．ディスクのドライブ名は，パソピアでは数字になっていますが，CP/Mでは，英字を使うことになっているので，1→A，2→Bということになります．また，現在アクセスしているディスク(ログインディスクといいます)の上のファイルを指定する際は，このうち〝ドライブ名：〟の部分は省略できます．このログインディスクのドライブ名は，つねに〝>〟の前に表示されていて，これを切り替えるには，ドライブ名：だけを入力します．つまり，〝A>〟と表示の出ているところに，〝B：〟と入力すれば，ログインディスクはBにかわり，〝B>〟という表示が出るわけです．また，ファイル名というのは，〝.〟で区切られておりその前8字以内，後3文字以内が許される長さです．

この後3文字は，エクステンションと呼ばれファイルの性質を表し，ある種のファイルには決まったエクステンションを付ける必要があります．例えば，前に挙げた〝.COM〟がその例です．そして，このファイル名では，小文字も大文字に変換されて解釈されます．

次に，ファイル名指定の2種類の方法の説明にもどります．コマンドには，SAVEのように1つだけのファイル名を指定する必要があるものと(1つのファイルに，2つの名前を付けるわけにはいきません)，ERAのようにファイル名のグループを指定できると便利なものがあります．(つまり，複数のファイルを一度に抹消できるわけです)前者のような場合は，普通にファイル名を指定するわけですが(マニュアルでufnと書かれているものです)，後者の場合は，〝*〟とか〝?〟という記号を使うのです．(マニュアルでafnと書かれているものです)〝*〟は〝.〟を含まないあらゆる文字列に対応し，〝?〟1文字は〝.〟以外のあらゆる1文字に対応します．例えば，ERA *.COMとすれば，エクステンションがCOMであるようなファイル(1～8文字目は何でもかまいません)．が消去されるわけです．また，ファイル名を*.*とすれば，あらゆるファイルが対応することになります．

いくつかの例を次の写真で見て下さい．

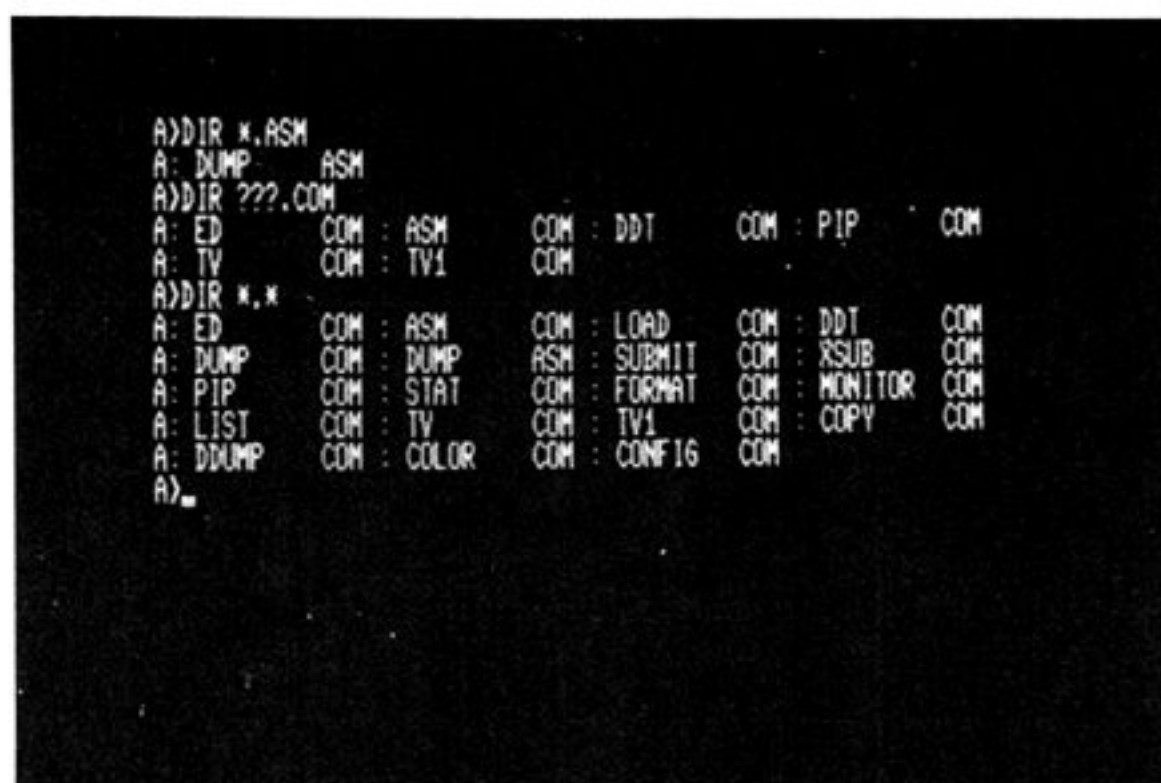



写真 9 - 2 - 1
DIRの実行

なお，SAVEの形式，SAVE　n ufuのnは，SAVEするページを示し，単位は，256バイト(＝ページ)です．TPAは，100H番地からはじまるので，このSAVEでセーブされるメモリも，100H番地からです．続くufnはファイルネームになります．

9 - 2 - 2　トランジェントコマンド

前節に続き，トランジェントコマンドの説明をします．なお，パソピアーCP/Mには，前に挙げたCOLORのように，標準CP/Mにないコマンドも加えられますが，これについては，9 - 3　パソピアーCP/Mの拡張コマンドで説明します．標準のCP/Mトランジェントコマンドを並べてみると，ASM，DDT，ED，LOAD，DUMP，SUBMIT，XSUB，PIP，STATの9つです．つまり，次の写真の，"COM"の付いたものです．順番に説明を加えていきましょう．なお，説明には略記法，ufn，afnを使います．

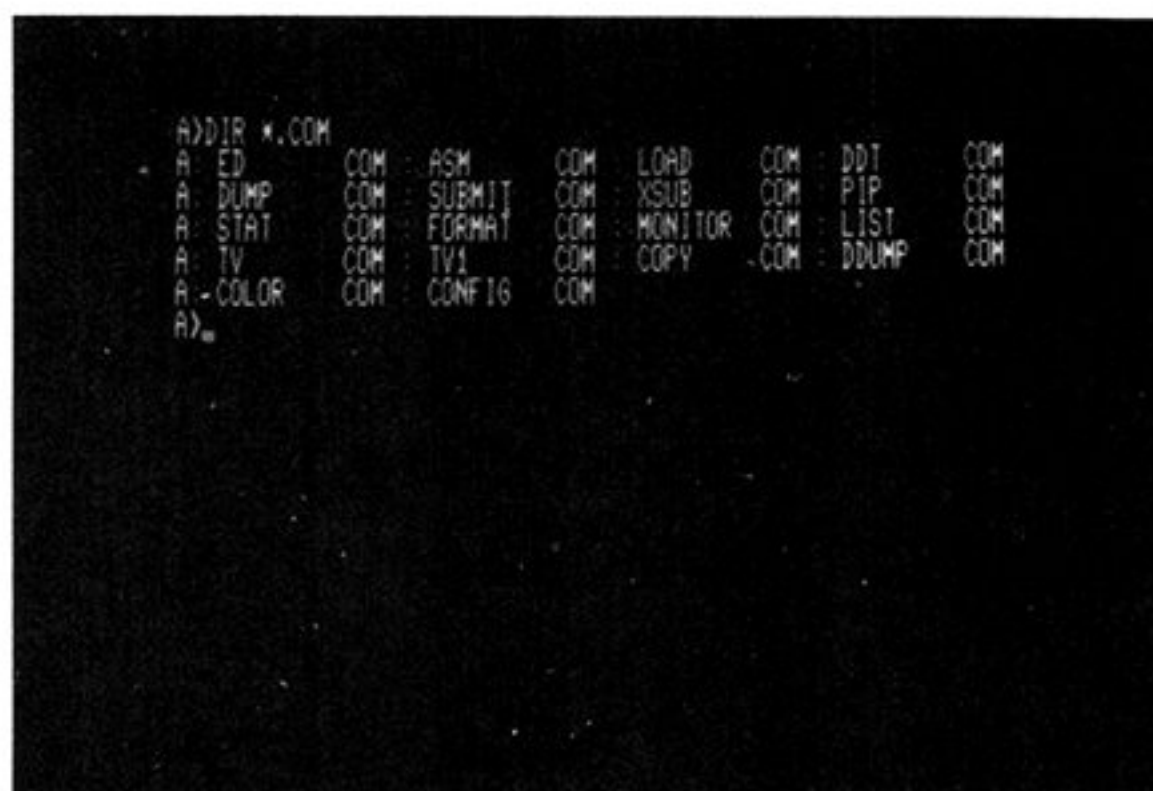



写真 9 - 2 - 2
DIR　*．COM

①　ED

EDとは，CP/Mのエディタで，テキストファイルをつくるのに用いられます．また，簡易英文ワードプロセッサとしても使えます．

EDは，メモリに入りきらないような巨大なファイルでもエディットできる機能や，ストリングのサーチ，変換機能など，大容量ファイルの編集能力があるのが特徴です．

まず，EDの起動ですが，ED　ファイル名とします．指定したフィイル名がディスク上になければ，そのファイルを作ります．その後のEDの働きは，テキストをメモリ上のエディット・バッファに取り込みます．そして，エディット途中で，ディスクとやりとりするときには，ディスク上のテンポラリファイル(エクステンションは$$$)に書き込まれます．そしてエディットが終了すると，すべてがテンポラリファイルに書き込まれ，テンポラリファイルのエクステンションが$$$からもとのものに変わり，また，オリジナル・ファイルの方は，エクステンションが，BAKに変わります．(ただちに抹消しないで，バックアップ・ファイルとして残すわけです)．エディット・バッファに入りきらないような巨大なテキストでも，少しずつロードしては，テンポラリバッファに書き込んでゆけば，エディットすることができるのです．

EDには，多くの機能があるため，残念ながら詳しいことは省略させていただきます．

② ASM

ASMとは，CP/Mで走る8080の機械語のアセンブラです．パソピアのCPUは，Z80ですがCP/Mは，8080用に開発されたものであるため，Z80特有の機械語はアセンブルできません．また，ニーモニックもZilog型式でなく，Intel型式で書かなくてはいけません．しかし，Z80の機械語は，8080に対し上位コンパチになっていますから，パソピアでも，十分強力な機械語開発ツールになるといえるでしょう．

③ DDT

DDTとは，Dynamic Debugging Toolの頭文字をつなげたものであり，8080機械語用の強力なデバッガです．T-BASICにモニタのないパソピアにはとても便利なものです．その形式は，単にDDTとするか〝DDT　ファイル名〟とします．前者の場合は，すぐにDDTのコマンド待ちになり，後者の場合は，ファイル名で指定されたプログラム(エクステンションは，HEXないしCOM)をロードした後，DDTのコマンド待ちとなります．

④ LOAD

トランジェントコマンドを解釈，実行するエクステンションCOM付きのファイルをつくるコマンドです．形式はLOAD　ufnですが，指定の際には，エクステンションは付けません．また，エクステンションHEXの付いたファイルは，②のASMによって作られます．つまり，トランジェントコマンドを新たに作るには，当然機械語で書かなくてはならないため，ソースリストからASMでエクステンションHEX付きのファイルをつくります．さらに，それにLOADコマンドを施して，エクステンションCOM付きのファイルをつくる，という手順を踏みます．例えばABC．HEXというファイルがあるとき，LOAD　ABCとすれば，ABC．COMというファイルが新たにでき，以後ABCとするだけで，他のコマンドと同様に使えることになります．

⑤ DUMP

ディスク上のファイルの内容を，1バイトずつ16進数で表示するもので，形式は，DUMP　ufn

です．これは，どのようなファイルでもダンプできるので，エクステンションまで含んだファイル名が必要です．なお，このDUMPは，トランジェントコマンドを実行するDUMP．COMの他，DUMP．ASMのファイルが含まれています．(写真9－1－1を見て下さい)．このDUMP．ASMが，DUMPコマンドのソースリストで，これからASMによって，DUMP．HEXをつくり，次いでLOADによってDUMP．COMがつくられるわけです．DUMP．ASMは，TYPEコマンドで読むことができますから試してみて下さい(写真9－2－3)．

```
A>TYPE DUMP.ASM
;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
;
;       NON GRAPH_
```

写真9－2－3
TYPEの実行

⑥ STAT

現在のファイル，デバイス割り付けの状態を表示します．形式はいくつかありますが，単にSTATとするとそのドライブに対して，R/W(Read/Write)かR/O(Read/Only)のいずれかと，残りメモリがキロバイト単位で表示されます．また，STATの後に〝ドライブ名．：″を付けると，今の動作を指定したドライブ名について行います．STAT　afnとすると，あてはまるファイルについてレコード長，キロバイト単位の大きさ，16キロバイト単位の大きさ，ドライブ名，エクステンション付きのファイル名が表示されます．最後には，残りのキロバイト数も表示されます．次に，STAT　ufnの後に，$R/O，$SYS，$R/W，$DIRのいずれかを付けると，それぞれライトプロテクト，DIRからかくす，$R/Oの解除，$SYSの解除という働きをufnに対応するファイルに行います．また，STATコマンドのデバイス割り付けに関する働きには，STAT　VAL：，STAT　DEV：の2つの形式があり，前者で可能なデバイス，後者で現在割り付けられているデバイスを表示します．

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>STAT DEV:
CON: is TTY:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:

A>_
```

写真9－2－4
STAT

⑦　SUBMIT

これは，一連のコマンドをまとめて実行する（バッチ処理といいます）コマンドで形式は，SUBMIT ufn　parm#1，…parm#nです．SUBMITは，CP/Mのコマンドをまとめ，その一連の手順をプログラム化して実行するものです．BASICでいえば，今まで説明してきたコマンドの実行法がダイレクト・ステートメントに相当し，SUBMITで実行するのがBASICプログラムとしての実行に相当するといえるでしょう．そのプログラムに相当するものは，①のEDでつくりますが，その際は行番号などはなくコマンドを順に並べるだけです．また，パラメータというのはファイル名などを変数にすることができ，DIR　$1などと書けるようになっています．その名前は，〝$〟に数字を付けて表すのですが，実行の際$1，…$nに，parm#1，…，parm#nの値が代入されてから実行されることになります．

⑧　XSUB

これは，SUBMITの拡張機能のためのコマンドで，SUBMITの中で呼び出す，エクステンションSUB付きのファイルの第１行におきます．そうすると通常のトランジェントコマンドのみならず，本来のトランジェントコマンドが実行中に，入力する命令までもファイルから読んで実行してくれます．つまり，ASM，DOT，EDなどのコマンドは，実行している途中にキーボードからいろいろなコマンドを受け付け，それに応じた動作をします．これは，その際に入力するコマンドまでもファイルから読み込むことを許可します．つまりトランジェントコマンドのレベルと，その内部でのコマンドのレベルで違いがあるのですが，それにかかわりなくファイルからの文字列を受け付けるようにするのが，XSUBということです．この機能は，CP/M２．２になって拡張されたものです．

⑨　PIP

PIPとは，ファイルの転送を行う命令であり，簡単にメディア変換が行えます．（写真９-２-５）

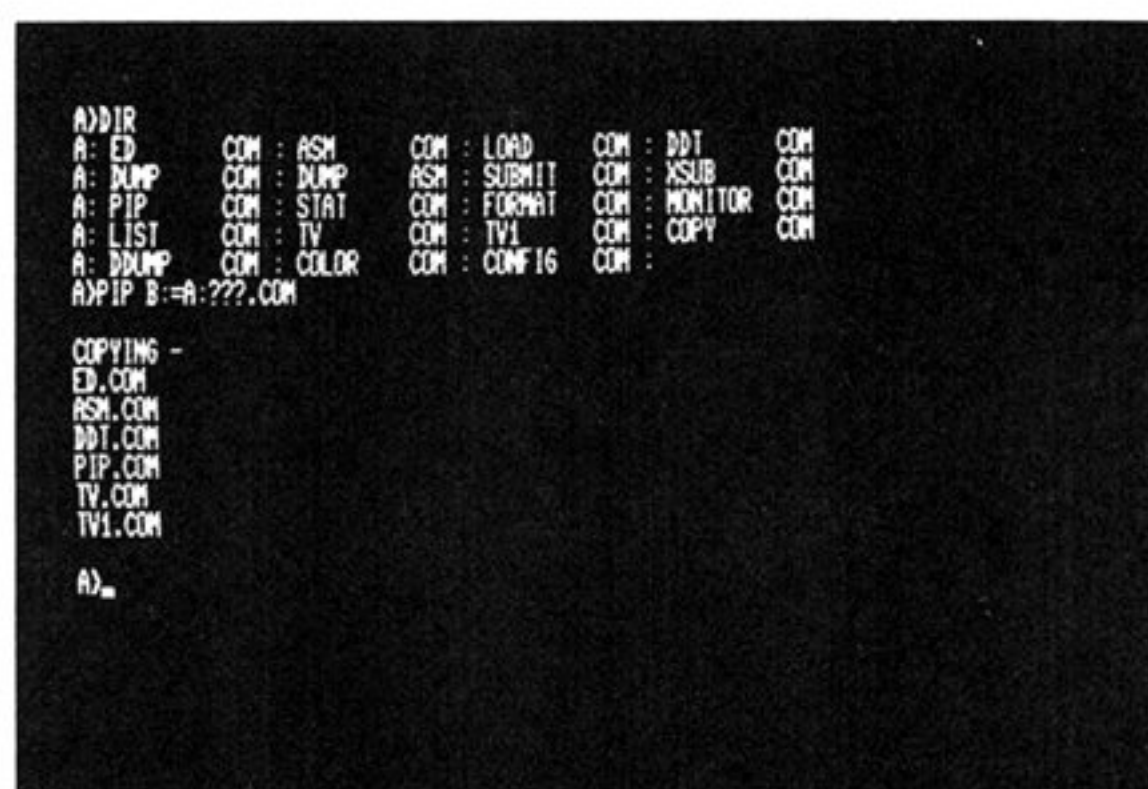



写真９-２-５
PIP

この形式は，いろいろありますが基本は，PIP ufn＝ufn1，ufn2，…，ufnnという形です．これは，ufn1，…ufnnのファイルを結合して，ufnという名前を付けることを意味します．n＝１ならば，単なるコピーということになります．また，転送のための省略形として，PIP　X：＝Y：afnという形も許されます．これは，afnにあてはまるファイルを，YからXへ転送することを意味し

ています．つまり，PIP　X：ufn＝y：は，PIP　X：ufn＝y：ufnと同じことです．単にPIPと入力すると，〝*〟とプロンプトが出るので，続けてコマンドライン(今までの例で〝PIP〟を除いた残り)を入力することもできます．また，PIPでは，ディスク間のみならずさまざまなデバイス間のデータ転送も可能です．それからコマンドラインの後に，PIPパラメータと呼ばれるものを[，]でくくって付けることにより，細かい制御が可能です．そのコードと意味は以下のとおりです．(nは数を表します)．

B　ブロックモード転送
Dn　頭からn文字をこえた部分を削除
E　転送操作のエコー表示(コンソール上)
F　フォームフィード(OCH)を削除
H　インテルHEX形式としてエラーチェック
I　Hの際に，ヌルコードを無視
L　大文字→小文字変換
N　行番号付加，N2とすると，数字の先頭のoも表示
O　転送の際，ファイル終了コード(1AH)を無視
Pn　n行ごとにページを送り，nが1またはないときは60行ごと
Q文字列ctrlZ
　文字列を発見すると，ファイル転送中止
S文字列ctrlZ
　文字列を発見すると，ファイル転送開始
Tn　タブスペースをnカラムに設定
U　小文字→大文字変換
V　リードアフターライトを行う
Z　入力時のパリティビットを0にする

9-3　パソピアCP／M特有のコマンド

パソピアCP/Mには，一般のCP/Mのコマンドのほかに，パソピア独自のコマンドがいくつかあります．ここでは，それについて順に説明してゆきましょう．

①　FORMAT

新しいディスクをフォーマットするコマンドです．〝FORMAT〟と入力して，画面の指示通りにすればよいので，何枚ものディスクを続けてフォーマットできます．

②　MONITOR

画面をクリアするコマンドです．単に〝MONITOR〟と入力します．

③ LIST

ディスク内のファイル名を表示します．DIRと違い，ファイル名，エクステンションの他，使っているキロバイト数が表示され，しかもアルファベット順にソートされて表示されます．また最後に，ファイル数，使用キロバイト数の合計，残りのキロバイト数が表示されます．DIRと同じく，ログインディスクを調べたいときには，単に〝LIST〞と入力し，他のディスクを参照したい場合は，〝ドライブ名：〞を付けます．

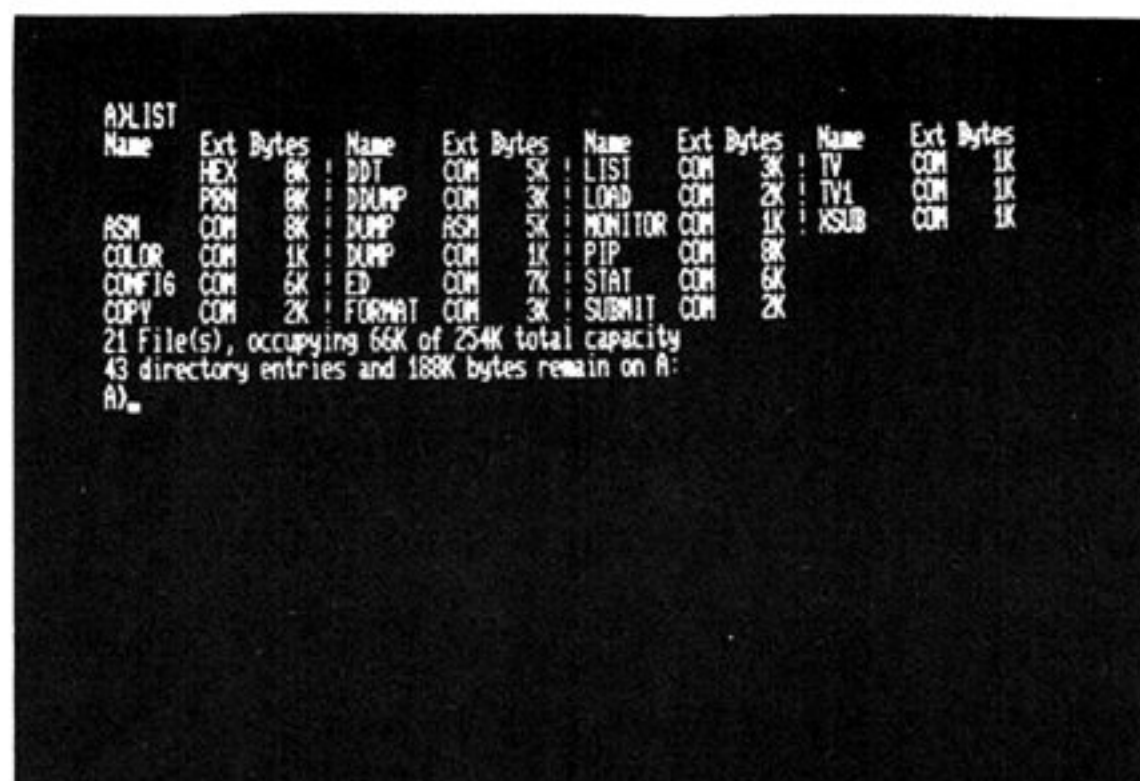



写真9-3-1
LIST

④ TV

画面のモードを36字×24行に設定するコマンドです．単に〝TV〞と入力します．次のTV1と同じく，DIRなどの画面表示が見にくくなってしまいます．

⑤ TV1

画面のモードを36字×19行に設定するコマンドです．単に〝TV1〞と入力します．

⑥ COPY

ディスクをコピーしてバックアップをとるコマンドです．〝COPY〞と入力した後，コピーされるディスクを，ドライブ1に，フォーマットされた新しいディスクをドライブ2に入れて，RETURNキーを押します．

⑦ DDUMP

ディスクの中の状態を直接16進数で表示するコマンドです．はじめに，ドライブ名，次いで，トラック，ヘッド，セクタの開始番号，表示するセクタ数，プリンタ出力の有無を入力します．表示は，アスキーダンプ付きで行われます．

⑧ COLOR

文字色，背景色を指定するコマンドです．BASICと同様，〝COLOR　5，1〞のように入力します．

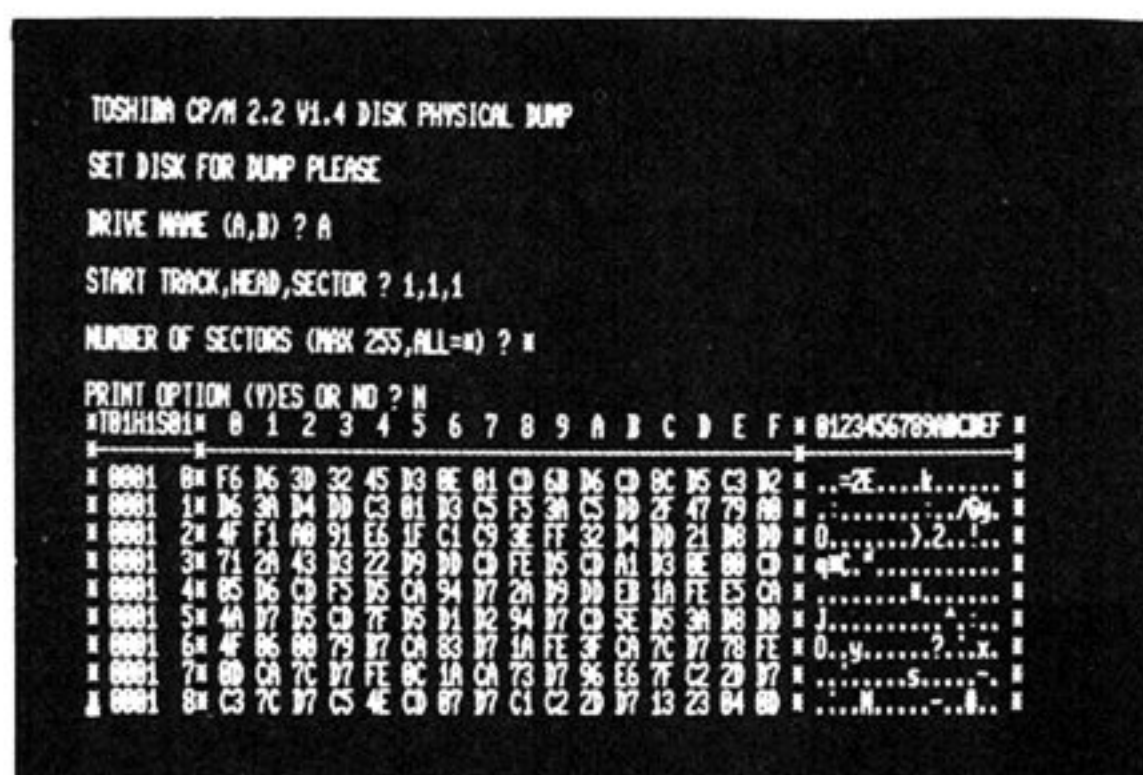



写真９－３－２
DDUMP

⑨　CONFIG

あらかじめ決められているさまざまな設定を変更するコマンドです．"CONFIG"と入力すると，少したって，メニューが出ます．CRTコントロールコード，プリンタコントロール，キーボードマトリックス，コミュニケーション・パラメータの変更，スクリーンサイズ(デフォルト値)の選択，SYSGEN(CP/Mのシステム部のコピー)などの機能があります．メニューの番号を入力して，それぞれの処理を画面の指示に従って行います．

APPENDIX

A.　T-BASIC　タイニ・モニタ
B.　T-BASIC　インタプリタ一覧表
C.　T-BASIC　ワーク・エリア一覧表
D.　T-BASIC　ジャンプテーブル一覧表
E.　T-BASIC　ROM 版 ver1.0と1.1の相違
F.　T-DISKBASIC ver2.0について
G.　I／O ポート一覧表
H.　MINI-PASCAL 内部ルーチン一覧表

A. T-BASIC タイニ・モニタ

以下に示すプログラムは機械語モニタです．このプログラムを使えば機械語のダンプやメモリの書き換えを行うことができます．

コマンドの説明

ESC　　機能のリストを出力します．

D　　メモリをアスキーダンプします．スタートアドレスとエンドアドレスを入力して下さい．チェックサム付で出力します．

S　　メモリに書き込みます．スタートアドレスを入力し，16進で打込んで下さい．－(マイナス)キーを押すとアドレスが1つ逆戻りし，RETURN を押すと1つ進みます．スペースを押すと書き込みを終了します．

P　　ダンプをプリンタに出力するかしないかのスイッチになっています．1回押すとON，もう1回押すと OFF で，ON のときはプリンタに出力します．

```
10000 ' xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
10010 ' xx                          xx
10020 ' xx  TINY MONITOR            xx
10030 ' xx                          xx
10040 ' xx          FOR T-BASIC     xx
10050 ' xx         VER. 1.0         xx
10060 ' xx                          xx
10070 ' xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
10080 WIDTH 80:KEY OFF:COLOR 5,0:CLS
10090 DEF FNF(X,Y)=VAL("&H"+LEFT$(HEX$(X),Y))
10100 ON ERROR GOTO 10910
10110 'CAPFLG=&HFE7C:POKE CAPFLG,1 :' Ver. 1.0        \
10120 'CAPFLG=&HFE7F:POKE CAPFLG,1 :' Ver. 1.1        | Ver. ニ ヨッテ カエルコト
10130  CAPFLG=&H1E3 :POKE CAPFLG,1 :' T-DISK BASIC   /
10140 ' ***   TITLE                ***
10150 PRINT"[[[  TINY MONITOR                          ]]]"
10160 PRINT"[[[                Ver. 1.0                ]]]"
10170 PRINT"[[[               Oct.'82                  ]]]"
10180 COLOR 7,0
10190 ' ***  GET COMMAND           ***
10200 WHILE INKEY$<>"":WEND
10210 POKE CAPFLG,255:IF DEV THEN PRINT">"; ELSE PRINT"]";
10220 A$=INPUT$(1):IF ASC(A$)<&H20 THEN A$="^"+CHR$(&H40+ASC(A$))
10230 PRINT A$;"   ";:CM=INSTR("DSWPJ",A$)
10240 IF A$="^[" THEN 10800
10250 IF A$="^B" THEN 10280
10260 IF CM=0 THEN PRINT"?":GOTO 10200
10270 ON CM GOTO 10290,10540,10600,10670,10700
10280 PRINT"Return to basic":ON ERROR GOTO 0:END
10290 ' ** Dump memory             ***
10300 PRINT"Dump memory":GOSUB 10760:GOSUB 10780
10310 PRINT:IF EAD=0 THEN 10200
10320 SADH=FNF(SAD,3):EADH=FNF(EAD,3):I=SADH
10330 ADR=I*16
10340 PRINT RIGHT$("000"+HEX$(ADR),4);" : ";
10350 IF DEV THEN LPRINT RIGHT$("000"+HEX$(ADR),4);" : ";
10360 FOR J=0 TO 7:M=PEEK(ADR+J):PRINT RIGHT$("0"+HEX$(M),2);" ";:NEXT:PRINT " "
;
10370 IF DEV THEN FOR K=0 TO 7:M=PEEK(ADR+J):LPRINT RIGHT$("0"+HEX$(M),2);" ";:N
EXT:LPRINT " ";
10380 FOR J=8 TO 15:M=PEEK(ADR+J):PRINT RIGHT$("0"+HEX$(M),2);" ";:NEXT:PRINT "
 ";
10390 IF DEV THEN FOR J=8 TO 15:M=PEEK(ADR+J):LPRINT RIGHT$("0"+HEX$(M),2);" ";:
NEXT:LPRINT "  ";
```

```
10400 SUM1=FNF(ADR,2)
10410 SUM2=ADR-SUM1*256
10420 SUM=SUM1+SUM2
10430 FOR J=0 TO 7 :M=PEEK(ADR+J):SUM=SUM+M:IF M>32 AND M<127 THEN PRINT CHR$(M)
; ELSE PRINT ".";
10440 IF DEV THEN IF M>33 AND M<&H7F THEN LPRINT CHR$(M); ELSE LPRINT ".";
10450 NEXT J:PRINT " "; :IF DEV THEN LPRINT" ";
10460 FOR J=8 TO 15:M=PEEK(ADR+J):SUM=SUM+M:IF M>32 AND M<127 AND M<>&H7F THEN P
RINT CHR$(M); ELSE PRINT ".";
10470 IF DEV THEN IF M>33 AND M<247 THEN LPRINT CHR$(M); ELSE LPRINT ".";
10480 NEXT J
10490 IF SUM>2^15 THEN SUM=SUM-2^15 ELSE IF SUM<-(2^15) THEN SUM=SUM+2^15
10500 PRINT" ";:PRINT RIGHT$("0"+HEX$(SUM),2):IF DEV THEN LPRINT" ";RIGHT$("0"+H
EX$(SUM),2)
10510 IF I+1<EADH THEN I=I+1:GOTO 10330
10520 IF DEV THEN LPRINT
10530 GOTO 10200
10540 ' **   Set memory        ***
10550 PRINT"Set memory":GOSUB 10760
10560 PRINT RIGHT$("000"+HEX$(SAD),4);":";RIGHT$("0"+HEX$(PEEK(SAD)),2);"-";
10570 A1$=INPUT$(1):IF A1$=CHR$(13) THEN SAD=SAD+1:PRINT"   ";:GOTO 10560 ELSE I
F A1$=" " THEN PRINT:GOTO 10200 ELSE IF A1$="-" THEN SAD=SAD-1:PRINT"   ";:GOTO
10560 ELSE PRINT A1$;
10580 A2$=INPUT$(1):IF A2$=CHR$(13) THEN SAD=SAD+1:PRINT"   ";:GOTO 10560 ELSE I
F A2$=" " THEN PRINT:GOTO 10200 ELSE IF A2$="-" THEN SAD=SAD-1:PRINT"   ";:GOTO
10560 ELSE PRINT A2$;" ";
10590 M=VAL("&H"+A1$+A2$):POKE SAD,M:SAD=SAD+1:GOTO 10560
10600 ' **        Ward serch       ***
10610 PRINT"Ward serch":INPUT"Serch";A$:GOSUB 10760:GOSUB 10780
10620 LENGTH=LEN(A$)¥2:IF LENGTH=0 THEN 10200 ELSE IF LENGTH>10 THEN PRINT"Too l
ong":GOTO 10200
10630 FOR I=1 TO LENGTH:WARD(I)=VAL("&H"+MID$(A$,I*2-1,2)):NEXT
10640 FOR I=SAD TO EAD:J=0:WHILE WARD(J+1)=PEEK(I+J) AND J<LENGTH:J=J+1:WEND
10650 IF J=LENGTH AND PEEK(I+LENGTH-1)=WARD(LENGTH) THEN PRINT RIGHT$("000"+HEX$
(I),4);"  ";
10660 NEXT:PRINT"Serch end":GOTO 10200
10670 ' **        Printer switch   ***
10680 IF DEV=0 THEN DEV=-1:PRINT"ON":GOTO 10200
10690                DEV=0  :PRINT"OFF":GOTO 10200
10700 ' ** JUMP                    ***
10710 PRINT"SUBROUTINE JUMP"
10720 GOSUB 10760
10730 INPUT" OK? (Y/N)";A$:IF A$="Y" OR A$="y" THEN CALL SAD
10740 GOTO 10200
10750 ' **    ADDRESS INPUT       ***
10760 INPUT"START ADDRESS";AD$:SAD=VAL("&H"+AD$):IF SAD<0 THEN SAD=SAD+2^16
10770 RETURN
10780 INPUT"  END ADDRESS";AD$:EAD=VAL("&H"+AD$):IF EAD<0 THEN EAD=EAD+2^16
10790 RETURN
10800 ' **  KEY FUNCTION          ***
10810 PRINT"***  KEY FUNCTION           ***
10820 PRINT
10830 PRINT" D      : DUMP MEMORY
10840 PRINT" P      : PRINTER SWITCH
10850 PRINT" S      : SET MEMORY
10860 PRINT" W      : WORD SERCH
10870 PRINT" J      : CALL SUBROUTINE
10880 PRINT" CTRL-B : RETURN BASIC
10890 PRINT
10900 GOTO 10200
10910 ' **  ERROR RESUME          ***
10920 PRINT" ?"
10930 RESUME 10190
10940 '------------------------------------------------------------------------
```

B. T-BASIC インタプリター覧表

アドレス	PASOPIA T-BASIC ROM内ルーチン	備考
0000	リセット	
0004	スタックポインタ番地	(F7FE)
0008	文字チェック	RST1の後の1バイトとテキストの文字の比較
0010	1文字読みこみ	RST2
0018	各デバイスへ1文字出力	RST3 (FE14により，各デバイスへ)
0020	HL，DE比較	RST4 (C，Zフラグをたてる)
0028	FAC符号チェック	RST5 (0，+，− →Acc=0，1，−1)
002B	V-RAM出力	
0030	FAC型チェック	RST6 整数 →C，NZ，M 文字 →C，Z，P 単精度→C，NZ，P 倍精度→NC，NZ，P
0033	V-RAM出力(1バイト)	
0038	ユーザーに開放	RST7，飛び先はFFF1
003B	FFEB〜に転送(データ)	入出力ルーチンの飛び先など
0050	FD00〜に転送(データ)	ワークエリア，ファンクションキーなど
01F0	リセット処理	
0600	キーボードキューリセット	
060C	1文字キー入力(待ちなし)	AccにASCIIコードが入る
065D	1文字キー入力(待ちあり)	Acc に ASCII コードが入る
069E	プリンタ1文字出力	Acc に ASCII コードを入れる

0 6 E E	画面コピー	
0 8 9 2	画面1文字出力	
0 8 C B	CLS エントリ	
0 8 D B	V-RAM文字読み	
0 8 E 7	V-RAM1文字表示	
0 8 F 4	カーソル位置セット	
0 9 3 6	カーソル off	
0 9 3 A	カーソル on／off	
0 A 6 4	V-RAMアドレス変換	
0 A C 5	V-RAM出力	
0 B 2 7	V-RAM入力(1バイト)	
0 F 9 A	LOCATE エントリ	
0 F B E	表示文字数(1行分)セット	
1 0 2 2	スクリーンモード設定	0，1，2に応じて，0，40H，80Hを入力
1 0 6 C	COLOR エントリ	
1 3 F 9	BLOAD#-1	
1 4 3 6	BSAVE#-1	
1 5 F 5	MOTOR エントリ	
1 8 1 3	音発生	音程Acc：0～107 長さDE：1～FFFF
1 8 6 B	周波数テーブル	
1 8 A 2	コマンド処理先アドレス	END(4C91)～TERM(547E)
1 9 4 E	関数処理先アドレス	LEFT$(639F)～MKD$(40EB)
1 9 B 8	中間言語テーブル辞書	A(19EC)～Z(1C8B)

19EC	中間言語対応表	AUTO(AA)~XOR(F6)
1C8C	特殊記号中間言語対応表	
1CAD	アドレス・テーブル	CDBL
1CB1		CINT
1CB3		文字型チェック
1CB5		CSNG
1CB7	数値演算ルーチンアドレス表	順に，倍精度：加，減，乗，除，比較，単精度：加，減，乗，除，比較，整数：加，減，乗，除，比較
1CD5	エラーメッセージ	エラーコード1(NF)~72(DO)(32~49は無し)
1D41	F800~に転送(データ)	
1D9C	"Error"	文字列データ
1DA3	"in"	〃
1DA8	"Ok"	〃
1DAD	"Break"	〃
1DB3	中間言語をスタック内から読む	
1DE2	END 実行	
1DFD	SN エラー	BASIC コマンド待ちへ
1E00	／0 エラー	〃
1E03	NF エラー	〃
1E06	DO エラー	〃
1E09	UF エラー	〃
1E0C	RWエラー	〃
1E0F	OV エラー	〃
1E12	MO エラー	〃
1E15	TM エラー	〃

1E17	エラーメッセージ出力	エラーコードはEレジスタに入れる
1EC6	BASIC コマンド入力	
1FB0	リンクポインタセット	
1FD6	LIST などの開始行セット	
1FE5	LIST などの終了行セット	
1FF2	行のサーチ	
2014	中間コード変換	インプットバッファを 中間コードバッファへ
21BC	LIST 定数保存	
2281	スペースのスキップ	
2291	FOR エントリ	
23C3	1文字読みこみ	
2442	値読みこみ	
2479	DEFSTR エントリ	
247C	DEFINT エントリ	
247F	DEFSNG エントリ	
2482	DEFDBL エントリ	
24BB	FC エラー	
24C0	開始行番号読みこみ	
24CA	行番号読みこみ	
24E2	HLの値を文字列に変換	文字列はDEで示されるアドレスから入る
24FF	RUN エントリ	
2513	GOSUB エントリ	
252A	GOTO エントリ	
255E	UL エラー	

2563	RETURN エントリ	
2580	REM エントリ ELSE エントリ COMMON エントリ DATA エントリ	共通
25A3	LET エントリ	
260E	ON エントリ	
2652	RESUME エントリ	
268A	ERROR エントリ	
2695	AUTO エントリ	
26C5	IF エントリ	
26FD	LPRINT エントリ	
2705	PRINT エントリ	
27C5	TAB, SPC(エントリ	
281A	SPC 処理	
2848	LINE エントリ	
284D	LINE　INPUT 処理	
287C	"? Redo from start"	文字列データ
28BC	INPUT エントリ	
297C	READ エントリ	
2A21	式評価	FAC にしまう
2B89	整数除算	
2B98	因存の値計算	
2C38	パラメータ読みこみ	
2C5A	小文字→大文字変換	
2C7A	16進文字列→FAC, DE	HL＋1からの文字列を変換

2CA1	8進文字列→FAC，DE	HL＋1からの文字列を変換
2D67	FAC 型チェック	RST6の飛び先
2D8C	OR	
2D96	AND	
2DA0	XOR	
2DAA	EQV	
2DB6	INP	
2DC6	LPOS エントリ	
2DC8	POS エントリ	
2DD4	USR エントリ	
2DF3	USR のサブルーチン	
2E0D	DEFUSR エントリ	
2E1C	DEF エントリ	
2E3F	FN エントリ	
2F8F	FN のサブルーチン	
2F99	DEFFN のサブルーチン	
2FA7	関数名読みこみ	
2FCC	INP エントリ	
2FE1	OUT エントリ	
2FE7	WAIT エントリ	
3001	WIDTH エントリ	
301C	表示文字数セット（1行分）	
3042	文字列を数値化→FAC	HLからの文字列
3073	LLIST エントリ	
3078	LIST エントリ	

30C2	1行出力	
30CB	INPUT バッファに入れる	
3175	LIST の際のデータ処理	
317F	LIST の際のREM処理	
3189	キーワード出力	LIST の内部処理
3227	定数出力	〃
3255	&H, &Oの出力	〃
32A8	DELETE エントリ	
32DC	REEK エントリ	
32E3	POKE エントリ	
32F9	HLからの文字列を数値化→HL	
3313	RENUM エントリ	
3435	OPTION エントリ	
3474	RANDOMIZE	
39ED	NAME エントリ	
3A38	OPEN エントリ	
3C63	KILL エントリ	
3C77	LOAD エントリ	
3C78	MERGE エントリ	
3CBA	SAVE エントリ	
3F7B	FIELD エントリ	
3FC0	SET エントリ	
40E5	MKI$ エントリ	
40E8	MKI$ エントリ	
40EB	MKD$ エントリ	

4 0 F E	CVI エントリ	
4 1 0 1	CVS エントリ	
4 1 0 4	CDBL エントリ	
4 1 4 8	CLOSE エントリ	
4 1 9 B	LFILES エントリ	
4 1 A 0	FILES エントリ	
4 2 7 0	PUT エントリ	
4 2 7 1	GET エントリ	
4 4 5 6	DSKO$ エントリ	
4 8 A E	DSKF エントリ	
4 8 D 7	LOC エントリ	
4 8 F 9	LOF エントリ	
4 9 0 E	EOF エントリ	
4 9 3 4	FPOS エントリ	
4 A A 2	BSAVE エントリ	
4 A D 8	BLOAD エントリ	
4 B 6 C	Out of memory のチェック	
4 B A C	NEW エントリ	
4 C 5 B	H L、D E比較	R S T 4
4 C 6 1	テキストの1バイトチェック	
4 C 7 3	RESTORE エントリ	
4 C 8 D	STOP エントリ	
4 C 9 1	END エントリ	
4 C E 2	CONT エントリ	
4 C F 7	TRON エントリ	

4CF8	TROFF エントリ	
4CFD	SWAP エントリ	
4D3B	ERASE エントリ	
4DB4	英字チェック	英字→NC，他→C
4DBF	CLEAR エントリ	
4E3E	NEXT エントリ	
4EF3	アドレス・テーブル	
4F71	スクロール・アップ	
5002	リンクポインタ上へ	
5015	リンクポインタ下へ	
5028	リンクの読みこみ	
5035	リンクセット	
504F	スクリーンエディタ	
5058	LINE INPUT("?" つき)	
505E	LINE INPUT("?" なし)	入力した文字列は インプットバッファ(F9C4～)へ
50F5	コントロールキーコード	
50FE	コントロールキー処理アドレス	
513F	LF処理	コントロールキーの処理
5143	CR処理	〃
51AD	ブレーク処理	〃
51BE	↓処理	〃
51D9	インサート処理	〃
5218	デリート処理	〃
526A	イレース処理	〃
5284	次の語への移動処理	〃

5 2 A 6	前の語への移動処理	コントロールキーの処理
5 3 0 5	インサートのサブルーチン	〃
5 3 1 F	デリートのサブルーチン	
5 3 3 F	インサートのサブルーチン	〃
5 3 5 7	スクロール・ダウン	
5 3 9 7	文字チェック	
5 3 E 9	EDIT エントリ	
5 4 5 8	SOUND エントリ	
5 4 7 E	TERM エントリ	
5 4 C 4	SGN エントリ	
5 6 1 7	CSAVE エントリ	
5 6 2 F	CLOAD エントリ	
5 6 B 5	ファイル名セット	
5 7 B A	PRESET エントリ	
5 7 B F	PSET エントリ	
5 8 4 9	LINE 処理	
5 9 5 B	PAINT エントリ	
5 A A 3	CIRCLE エントリ	
5 C F 0	WHILE エントリ	
5 D 0 F	WEND エントリ	
5 D 7 D	CALL エントリ	
5 D F 0	CHAIN エントリ	
6 0 A 1	COMMON エントリ	
6 0 A 4	WRITE エントリ	
6 1 1 D	OCT$ エントリ	
6 1 2 2	HEX$ エントリ	

6127	STR$ エントリ	
61AC	メッセージ出力	HLからからの文字列(最後は00)の出力
61C2	スペースをつくる	
61DD	ガベージ・コレクション	
633D	LEN エントリ	
6349	ASC エントリ	
6359	CHR$ エントリ	
6367	STRING$ エントリ	
6386	SPACE$ エントリ	
639F	LEFT$ エントリ	
63CF	RIGHT$ エントリ	
63D8	MID$ エントリ	
63F9	VAL エントリ	
64CC	INSTR エントリ	
64A0	MID$(コマンド)処理	
6515	パラメータの読みこみ	
6523	FRE エントリ	
653A	単精度減算	FAC=(HLからの4バイト)−FAC
653F	単精度加算	FAC=(HLからの4バイト)+FAC
6542	単精度減算	FAC=FAC−EDCBレジスタ
6545	単精度加算	FAC=FAC+EDCBレジスタ
6594	単精度正視化	
65A7	FACに0を代入	
65E7	FACを1ふやす	
6600	EDCBレジスタの符号反転	
6614	EDCBレジスタのシフト	

6658	LOG 係数データ	
6669	LOG エントリ	FAC=LOG(FAC)
66AC	単精度乗算	FAC=FAC＊EDCBレジスタ
670D	単精度除算	FAC=FAC／EDCBレジスタ
67D6	FAC符号チェック	
67F8	ABS エントリ	FAC=ABS(FAC)
67FC	符号反転	FAC=－FAC
6803	FAC符号反転	FAC=－FAC
680B	SGN エントリ	FAC=SGN(FAC)
680E	8ビット整数→16ビット整数	Acc→HL
6825	FACを PUSH	POP BC, POP DE とすれば, EDCBレジスタに入る
6832	HL からの4バイト→EDCB レジスタ→FAC	
6840	FAC→EDCBレジスタ	
684C	FAC→HLからの4バイト	
6853	HLから→DEから	FACの型のバイト分(変数値転送)
6854	DEから→HLから	
685F	MSB セット	
6874	倍精度FAC値とりだし	FC4Dから→FACへ
687C	倍精度FAC値保存	FAC→FC4Dから
688C	単精度比較	
68B7	整数比較	
68F1	倍精度比較	
68F8	CINT エントリ	
696C	CSNG エントリ	
6996	CDBL エントリ	
69AE	文字型チェック	違えば、タイプミスマッチ

69DE	FIX エントリ	FAC＝FIX(FAC)
69ED	INT エントリ	FAC＝INT(FAC)
6A6A	配列添数読みこみ	
6A87	整数減算	FAC＝HL－DE
6A92	整数加算	FAC＝HL＋DE
6AB2	整数乗算	FAC＝HL＊DE
6B03	整数除算	FAC＝HL／DE
6B40	HLの絶対値をとる	HL＝ABS(HL)
6B45	HLの符号反転	HL＝－HL
6B64	MOD エントリ	
6B75	倍精度減算	FAC＝FAC－(FC4Dから)
6B7C	倍精度加算	FAC＝FAC＋(FC4Dから)
6BE1	倍精度正視化	
6C5F	倍精度符号反転	FAC＝－FAC
6CA8	倍精度乗算	FAC＝FAC＊(FC4Dから)
6D7B	倍精度除算	FAC＝FAC／(FC4Dから)
6E04	文字列→数値	HLからの文字列→FAC
6E0B	文字列→数値	
7014	HLの値出力	
7022	数値→文字列	FAC→(FC56から)
745B	整数FAC→文字列	左の0はつめない
7504	数値データ	順に10000, 1000, 100, 10, 1
750D	FAC→8進数文字列	
7511	FAC→16進数文字列	
7557	SQR エントリ	FAC＝SQR(FAC)
75BD	EXP エントリ	FAC＝EXP(FAC)

760A	EXP 係数データ	
7627	べき級数計算ルーチン①	奇数次数のみからなるもの
7636	べき級数計算ルーチン②	
7609	RND ルーチン	FAC=RND(FAC)
76DA	COS エントリ	FAC=COS(FAC)
76E0	SIN エントリ	FAC=SIN(FAC)
7756	SIN 係数データ	
777B	TAN エントリ	FAC=TAN(FAC)
7790	ATN エントリ	FAC=ATN(FAC)
77B4	ATN 係数データ	
77DE	DIM エントリ	
77E3	変数値読みこみ	
78D7	新変数作成	
793B	未定義変数のわりだし	
794E	配列読みこみ	
79FE	BS エラー	
7AEB	USING エントリ	
7C98	各デバイスへの出力	RST3でここへ飛ぶ
7CFC	ラインプリンタリセット	
7DCD	CR出力	
7E0A	INKEY$ エントリ	
7E2B	コールド・スタートの処理の続き	
7F5F	最初のメッセージ	

C. T-BASIC ワーク・エリアー覧表

アドレス	内容
F800 ~0D	除算用サブルーチン
F80E ~30	RND用ワーク
F831 ~34	前回のRNDの値
F835 ~48	USRO~USR9 の処理先アドレス
F849	エラーコード
F84B	プリンタヘッド位置
F84C	1のとき出力をプリンタへ
F84F	スクリーン横文字数
F852	スクリーン不表示フラグ
F853 ~4	スタックのアドレス保存
F855 ~6	実行中の行番号
F857 ~8	BASIC テキスト開始アドレス
F885 ~F9C3	中間言語バッファ
F9C4 ~FAC6	キー入力バッファ
FAC7	DIM のルーチンから CALL したフラグ
FAC8	FAC型(整数→2~文字→3~単精度→4~倍精度→8)
FAC9	中間言語~リストの変換フラグ
FACB ~C	ポインタセーブアドレス
FACD	Acc セーブアドレス
FACF	フローティングポイントレジスタ
FAD7 ~8	フリーエリア最終アドレス
FADB	ストリングスタック
FAEA	暗黙の変数型(A~Z26字分~最初はすべて4)
FAF9	ストリングディスクリプタ
FAFC ~D	ストリング~フリースペース開始アドレス
FB02 ~3	FOR のテキストポインタセーブ
FB06	ルーチン認識コード
FB0A	行番号→アドレス変換フラグ
FB10 ~1	テキスト・ポインタ
FB14 ~5	ERL
FB16 ~7	"." で表わされる行番号
FB18 ~9	エラー文のアドレス
FB1A ~B	on error goto の飛び先アドレス

FB1C	on error goto のフラグ
FB1D ~E	HLのセーブ
FB1F ~20	ブレーク時の行番号
FB21 ~2	ブレーク時アドレス
FB23 ~4	変数エリア開始アドレス
FB25 ~6	配列エリア開始アドレス
FB27 ~8	スタック開始アドレス
FB29 ~A	READ 文ポインタ
FC36	SWAP 用変数バッファ
FC3E	TRON フラグ
FC40	FAC
FC4D	倍精度演算用 FAC
FC56	数値から変換された文字列
FCB4	BSAVE 最終アドレス+1
FD39	スクロール開始行+1
FD3E ~F	カーソル位置
FD5C	カーソル on/off フラグ
FD72	PFキー内容
FE08	20Hに out したもの
FE09	PIO port A カレントステータス

FE0D ~12	ファイル名
FE14	出力デバイスフラグ 0→CRT, MSB 0→LP~MSB 1→CMT
FE28 ~9	画面の LINE INPUT のときのカーソル初期位置
FE2A	X座標をどこまで読むか
FE7D	カーソルX座標
FE7E	カナモードフラグ 1……カナ
FE7F	CAPS フラグ 1……大文字
POKE &hfe75, 1	
FE97	BLOAD アドレス指定フラグ
FE9E ~F	サウンド長カウンタ
FEF0	割込ベクタ
FF01	キーボードキューバッファ
FF21	通信(−3)受信キュー
FF61	通信(−4)受信キュー
FFEB	通信(−3)モード
FFEC	通信(−3)速度
FFEE	割込タイマ
FFF0	プリンタ監視時間
FFF1	RST 7 (初めはJP FFF1の永久ループ)
FFF4	1文字プリンタ出力ルーチンへ
FFF7	画面コピールーチンへ
FFFA	1文字キー入力ルーチンへ(待ちなし)
FFFD	画面1字出力ルーチンへ

D. T-BASIC ジャンプテーブル一覧表

キーワード	中間コード	ROM版 ver 1.0	ROM版 ver 1.1	Disk版 ver 1.0
AUTO	AA	263D	2695	3540
AND	F4			
ABS	FF86	6710	69ED	7DCE
ATN	FF8E	76A8	777B	
ASC	FF95	6261	63F9	7916
ATTR$	E8			
BSAVE	D2	49C6	4AA2	59A8
BLOAD	D1	49FC	4AD8	59DE
CLOSE	BF	40B5	4148	501F
CONT	99	4C00	4CE2	5CE6
CLEAR	92	4CDD	4DBF	5DC3
CLOAD	9B	5540	562F	695E
CSAVE	9A	5528	5617	6946
CSRLIN	E7			
CINT	FF9D	6810	1DFD	7ECE
CSNG	FF9E	6884	69F8	7F42
CDBL	FF9F	68AE	696C	7F6C
CVI	FFA1	406B	69DE	4FD5
CVS	FFA2	406E	40FE	4FD8
CVD	FFA3	4071	4101	4FDB
COS	FF8C	75F2	75BD	
CHR$	FF96	6271	6349	7926
CALL	B3	5C97	5D7D	7351
COMMON	B5	2526	257E	33F9
CHAIN	B6	5D0A	5DF0	73C4
COM	FFD4	****	****	
CIRCLE	CA	59BD	5AA3	6DD2
COLOR	CF	103D	106C	13C5
CLS	9F	08A9	08C8	0940
DELETE	A9	3247	32A8	4182
DATA	84	2526	257E	33F9
DIM	86	76F6	77DE	
DEFSTR	AC	2421	2479	32C0
DEFINT	AD	2424	247C	32C3
DEFSNG	AE	2427	247F	32C6
DEFDBL	AF	242A	2482	32C9
DSKO$	B9	43A0	4456	5351
DEF	97	2DBB	2E1C	3CFC
DSKI$	E9A4	47F6		
DSKF	FFA4	47F6	4104	57AF
DRAW	D5	****	****	7166
ELSE	A1	2528	2580	33FB
END	81	4BAF	4C91	5C95
ERASE	A5	4C59	4D3B	5D3F
EDIT	A6	52FF	53E9	63ED
ERROR	A7	2632	268A	3535
ERL	E08B	74D5		
ERR	E1A5	483D		
EXP	FF8B	74D5	6669	
EOF	FFA5	483D	48AE	580F
EQV	F7	3F1B		
FOR	82	2239	2291	30CC
FIELD	BB	3F1B	3F7B	4E57
FILES	C28F	410D	41A0	507C
FN	DDA0	68F6		
FRE	FF8F	643B	7790	7AF0
FIX	FFA0	68F6	6996	7FB4
FPOS	FFA8	4859	48F9	5835
GOTO	89	24D2	252A	338C
GO TO	89	24D2	252A	338C
GOSUB	8D9A	24BB	2513	335A
GET	BC	41DE	4371	101B
HEX$	FF9A	603A	611D	76EF
INPUT	85	285B	28BC	378E
IF	8B85	266D	26C5	3570
INSTR	E490	2F6B		
INT	FF85	6905	680B	7FC3
INP	FF90	2F6B	6523	3EAC
IMP	F8D3			
INIT	D4	****	****	1598
INKEY$	EB	26A5		
KEY	FFD3	3012		
KILL	C49B	3C07	3C63	4B40
KANJI	FFD5	****	****	
LPRINT	9D	26A5	26FD	35A8
LLIST	9E	3012	3073	3F2F
LPOS	FF9B	2D65	6122	3CA6
LET	88	254B	25A3	341E
LOCATE	C9	0F78	0F9A	12F3
LINE	B0	27E7	2848	3713
LOAD	C08A	3C1B	3C77	4B54
LSET	C5A6	3E76	3ED6	4DB2
LIST	9392	3017	3078	3F34
LFILES	C881	4108	419B	5077
LOG	FF8A	6581	76E0	7C36
LOC	FFA6	4814	490E	57D8
LEN	FF92	6255	2DCB	790A
LEFT$	FF81	62B7	1DFD	796C
LOF	FFA7	482C	48D7	57FA
MOTOR	CBAA	15A7	15F5	1B11

MERGE	C1AB	3C1C	3C78	4B55
MOD	F983	62F0		
MKI$	FFA9	4052	4934	4FBC
MKS$	FFAA	4055	40E5	4FBF
MKD$	FFAB	4058	40E8	4FC2
MID$	FF83	62F0	63CF	79A5
NEXT	83	4D5C	4E3E	5E42
NAME	C3	3991	39ED	48CA
NEW	94	4AC9	4BAB	5AF0
NOT	DF			
OPEN	BA99	39DC	3A38	4915
OUT	9C	2F80	2FE1	3EC1
ON	95	25B6	260E	3483
OR	F5	26AD		
OCT$	FF99	6035	6386	76EA
OPTION	B7	33D4	3435	430F
OFF	EA91	2D6A		
PRINT	9197	26AD	2705	35B0
PUT	BD	41DD	4270	1018
POKE	98	3282	32E3	41BD
POS	FF91	2D6A	2FCC	3CAB
PEEK	FF97	327B	6359	41B6
PORT	FF9C	****	****	14BB
PSET	CD	56D9	57BF	6AEE
PRESET	CE	56D4	57BA	6AE9
POINT	FFD2	4B91		
PAINT	CC	5875	595B	6C8A
PLAY	D6	****	****	
RETURN	8E	250B	2563	33C5
READ	8782	291B	297C	384E
RUN	8A88	24A7	24FF	3346
RESTORE	8C	4B91	4C73	5C77
REN	8F	2528	2580	33FB
RESUME	A8D0	25FA	2652	34FD
RSET	C6	3E75	3ED5	4DB1
RIGHT$	FF82	62E7	639F	799C
RND	FF88	7581	7557	
RENUM	AB	32B2	3313	41ED
RANDOMIZE	B8	3411	3474	434D
SCREEN	FFD0			
STOP	9084	4BAB	4C8D	5C91
SWAP	A487	4C1B	4CFD	5D01
SET	BE89	3F60	3FC0	4E9C
SAVE	C793	3C5E	3CBA	4B97
SPC(	DE			

STEP	DB98	629E		
SGN	FF84	6723	63D8	7DE1
SQR	FF87	746F	67F8	
SIN	FF89	75F8	7669	
STR$	FF93	603F	633D	76F4
STRING$	E2			
SPACE$	FF98	629E	32DC	7953
SOUND	D08D	536E	5458	645C
THEN	D9	538F		
TRON	A2D1	4C15	4CF7	5CFB
TROFF	A3	4C16	4CF8	5CFC
TAB(	DA			
TO	D894	6311		
TAN	FF8D	7693	76DA	
TERM	D3	538F	547E	6557
TIME	FFD1	2F86		
USING	E3	5C0A		
USR	DC	5C29		
VAL	FF94	6311	6127	79C6
VARPTR	E6			
WIDTH	A0	2FA0	3001	3EE1
WAIT	96	2F86	2FE7	3EC7
WHILE	B1	5C0A	5CF0	72C4
WEND	B2	5C29	5D0F	72E3
WRITE	B4	5FB7	60A4	7671
OR	F6			
+	EF			
−	F0			
*	F1			
／	F2			
∧	F3			
¥	FA			
'	E5			
>	EC			
=	ED			
<	EE			

E. T-BASIC ROM 版 ver1.0と1.1の相違

ROM 版のT-BASIC は ver1.0とver1.1の2種類あり，ver1.1では次の点が変更されています．

○ INS DEL キーのセルフリピートはキーを離せば即時停止．

○スクリーンモード0（テキストモード）を使えるようにした．

○SOUND 文の休符指定（SOUND0, n）が使える．

○BLOAD で絶対アドレスの指定ができる．

○ファ音（SOUND機能）の周波数を修正．

○ファンクションキーの定義の引数に文字変数を使える．

○画面のスクロールアップのスピードはテキストモードで高速化される．また，画面消去も高速化される．

○ハードコピー時にドットプリンタII以外のプリンタを使ってハードコピーしたときに“16”の文字が印字されないようにした．（テキストモード）

現在流通している PA7010およびT-BASIC の ROM パックは ver1.1ですので，ver1.0のユーザはROM PACを使えば ver1.1の BASIC を使用できます．

なお，ver1.1で AUTO を使った場合，前から存在する行番号が発生したときに“＊”が表示されるのですが，RETURN のみを押した場合その行が削除されてしまうので注意して下さい．

F. T-DISKBASICver2.0について

東芝では，T-DISKBASIC の改良版として，ver2.0を出しました．

T-DISKBASIC ver2.0は ver1.0に対し次の点が変化しています．

○KANJI キーは ver1.0では無視されていたが，ver2.0では次のように出力される．

KANJI キーのみ押したとき──→ “&K” を出力．

SHIFT＋KANJI キー──→ “KANJI” を出力．

○拡張ユニット経由で8インチフロッピーディスクを使用することができる．

○ファイン・グラフィックモードでも LINE や PSET のカラーコードが有効になった．ただし描く図形の左側にカラー・アトリビュートを書込むスペースがなければ，左側のグラフィックの色になる．

○漢字 ROM PAC2をドライブ用のサポートルーテンなしで使用できる．

○プログラムを修正した結果メモリ容量が不足しても，正常に Out of memory を出力し，システムがクラッシュしないようになった．

○フリーエリアが約3キロバイト減少した．

○カセットの LOAD／SAVE の時間が短縮した．しかしカセットファイルの互換性は残した．

G. I／O ポート一覧表

ラベル名	ポート・アドレス		内　容
CRTDA		00H	：8255 PIO CH,A データポート(出力のみ)
	BIT7-0	〃	V-RAM アクセス 07-00
CRTDB		01H	：8255 PIO CHB データポート(出力のみ)
	BIT7-0	〃	V-RAM WRITE DATA
CRTDC		02H	：8255 PIO CH,C データポート(入力のみ)
	BIT7-0	〃	V-RAM READ DATA
CRTDS		03H	：8255 PIO コントロールポート
CRTIA		08H	：8255 PIO CH,A データポート(出力のみ)
	BIT7	〃	ファイングラフィックモードセット信号
	BIT6	〃	グラフィックモードセット信号
	BIT5	〃	WIDE(80カラム) CRTセット信号
	BIT4-3	〃	
	BIT2-0	〃	背景色セット信号(緑／赤／青)
			LCD(液晶)／BIT0 1：10ラスタ 0：8ラスタ
CRTIB		09H	：8255 PIO CH,C データポート(出力のみ)
	BIT7	〃	アトリビュート READ DATA
	BIT6	〃	CRT BUSY信号(水平同期)
	BIT5	〃	垂直同期 信号
	BIT4	〃	ディスプレイ タイプ(1：CRT 0：LCD)
	BIT3-0	〃	
CRTIC		0AH	：8255 PIO CH,C データポート(出力のみ)
	BIT7	〃	アトリビュート WRITE DATA
	BIT6	〃	V-RAM READ 信号(LOW-WRITE)
	BIT5～0	〃	V-RAM アドレス 13～08
CRTIS		OBH	：8255 PIO コントロールポート
CRTCA		10H	：CRTC(HD465055)レジスタアドレス(出力のみ)
CRTCD		11H	：CRTC データ ポート (入出力)
	R00	／	水平総大字数
	R01	／	水平表字文字数
	2	／	水平同期位置
	3	／	SYNC+WITH(VVVVHHHH)
	4	／	垂直総文字数
	5	／	トータル ラスタ アドジャスト
	6	／	垂直表示文字数
	7	／	垂直同期位置
	8	／	インタ・レース／スキュー(CCDD-VS)

	9　　／	MAX　ラスタ　アドレス
	10　　／	カーソル　スタートラスタ(−BPDDDDD)
	11　　／	ストップラスタ
	12　　／	スタートアドレス　HIGH
	13　　〃	LOW
	14　　／	カーソルアドレス　HIGH
	15　　〃	LOW
DFCDO	18H	:DATAファイルカセットポート #0
DFCP1	19H	:　〃 #1
DFCP2	1AH	:　〃 #2
DFCP3	1BH	:　〃 #3
DFUPO	1CH	:DATAファイルユニットポート #0
DFUP1	1DH	:　〃 #1
DFUP2	1EH	:　〃 #2
DFUP3	1FH	:　〃 #3
IOCA	20H	:8255PIO　CH,A　データポート(出力のみ)
	BIT7　〃	プリンタ　プライム信号
	〃 6　〃	プリンタ　ストローブ　〃
	〃 5　〃	ACMT　リモート　コントロール信号(Low)
	〃 4　〃	ACMT　WRITE　DATA
	〃 3-0　〃	
IOCB	21H	:8255PIO　CH,B　データポート(入力のみ)
	BIT7　〃	プリンタ　FAULT 信号(Low)
	〃 6　〃	BUSY　〃
	〃 5　〃	ACMT　READ-OOT　ATA
	〃 4　〃	SIO　CH/TXC(Low)
	〃 3　〃	DCD　(〃)
	〃 2　〃	DSR　(〃)
	〃 1　〃	CTS　(〃)
	〃 0　〃	RXD
IOCC	22H	:8255　PIO　CH,C　データポート
	BIT7　〃	SIO　ST1(Low-Out)
	〃 6　〃	DTR(　〃　)
	〃 5　〃	RTS(　〃　)
	〃 4　〃	TXD(Out)
	〃 3　〃	メモリモード　／0ROM／0RAM／1ROM
		PAC
	〃 2　〃	／0　／1　／0
	〃 1-0　〃	(No, IN)
IOCNT	23H	:8255　PIO　コントロールポート
	1DDDDDDD　／	モードコントロールワード

	0 0	／	グループA　モード0
	0 1	／	モード1
	1 X	／	モード2
	1	／	ポ ー ト A　INPUT
	0	／	OUTPUT
	1	／	ポートC(U)　INPUT
	0	／	OUTPUT
	1	／	グループB　モード1
	0	／	〃　0
	1	／	ポ ー ト B　INPUT
	0	／	OUTPUT
	1	／	ポ ー ト C　INPUT
	0	／	OUTPUT
IOCNT	OXXXDDDD		BIT　コントロールワード(ポートCのみ)
	BIT3-1	／	BIT　ナンバー
	BIT0	／	BIT　SET(1)／RESET(0)
CTC0	28H		：Z80ACTC　CH. 0　ACMT／RS232C　タイミングCNT.
CTC1	29H		：CH1.　スピーカ周波数
CTC2	2AH		：CH2.　KB　タイミングコントロール
CTC3	2BH		：CH3.　システムクロック(64Hz)
	DDDDDDD1		コントロールワード(タイマーモードのみ)
	BIT7	／	1　割込みイネーブル
	〃 6	／	モードタイマ(0)／カウンタ(1)
	〃 5	／	プリスケーラ256(1)／16(0)
	〃 4	／	EDGEセレクトfalling(0)／rising(1)
	〃 3	／	タイマトリガ
		／	0　オートマチックトリガ
		／	1　CLK／TRG+Starts Timer
	〃 2	／	Time constant fallows on 1
	〃 1	／	ソフトウェア　Reset
	DDDDDXX0		割込み　ベクトルワード
NMCNT	3CH		：メモリモードセレクト&リセット
	BIT7-3	〃	
	BIT2	／	ハードウェア Reset
	〃 1	／	RAM　Select
	0	〃	RAM　〃　(1：ROM　PAC-1　0：内部ROM)
KBAD	30H		：Z80A　PIO　CH.Aデータポート(出力のみ)
	BIT7	〃	スピーカのためのアラーム
	BIT6～4	〃	KBスキャンブロックのセレクト
	BIT3～0	〃	KBスキャンラインのセレクト
KBBD	31H		：Z80A　PIO　CH.B　データポート(入力のみ)
	BIT7～0	〃	KBスキャン　インプットデータ(Low Level)
KBAC	32H		：Z80A　PIO　CH.A　コントロールポート

	DD――1111	コントロールワード
	Mode	
	00 ／	MODE0 BYTE INPUT
	01	1 OUTPUT
	／	
	10	2 BIDIRECTIONAL
	／	
	11	3 BIT I／O
	／	
	DDDDDDD0	割込 ベクトルワード
	DDDDDDDD	I／O コントロール&マスクコントロール
	DDDD0A11	割込 コントロールワード
	BIT7 ／	割込 ENABLE(1)／DISABLE(0)
	BIT6 ／	AND(1)／OR(0) FUNCTION
	BIT5 ／	ACTIVE LEVEL HIGH(1)／LOW(0)
	BIT4 ／	マスクワードの Follow on 1
KBBC	EQU 33H	:Z80A PIO CH.B コントロールポート
TSPK1	40H	:KBCPU パネル&コントロール
	BIT7 〃	Test Sel. 1
	6 〃	〃 2
	5-4 〃	MBZ
	2 〃	WA／KB Sel
	1 〃	ERR STOP SW
	0 〃	START SW
	〃	
	7 〃	ERR Continue
	6 〃	END
	3 〃	NMI OUT
	2 〃	EXT INT・OUT
	1 〃	COMP／ACMT 信号 Sel
	0 〃	LCD／CRT Sel
TSPK2	41H	: Power Supply Voltz Information
	BIT5 〃	EXTM1信号
	4 〃	EXT 8MHz クロック
	3 〃	P12-12
	2 〃	M12-12
	1 〃	P12-5
	0 〃	M12-5
TSPK3	42H	:CRT Status far Read
	BIT5 〃	CLOCK 信号
	4 〃	N-VSYNC 〃
	3 〃	N-HSYNC 〃
	2 〃	VIDEO 緑信号
	1 〃	〃 赤 〃
	0 〃	〃 青 〃
TSPK4	43H	:LCD Status for Read
	BIT5 〃	CLOCK 信号
	4 〃	N-VSYNC 〃
	3 〃	N-HSYNC 〃

	2 〃	VIDEO 〃
	0 〃	SHIFT MODE(10ラスク)信号(low)
TSPK5	44H	:PROTD(PJ. 9, PJ10)Turn Back データ
TSPK1	48H	FDC
	BIT7-6, 4.	MBZ
	〃 5 〃	FDC WA TEST GO
	〃 3 〃	STEP AJUSTED
	〃 2 〃	STEP SWITCH
	〃 1 〃	ERROR STOP SWITCH
	〃 0 〃	START SWITCH
	BIT7-1 〃	SEST SEQ Number Indicate
TSPF2	49H	:アダプタコントロール
	BIT7 〃	ERR Continue
	〃 6 〃	END 〃
	〃 3 〃	VORITE PROTECT コントロール
	〃 2 〃	READ-DATA Select (1=OSC, 0=FDD)
	〃 1-0 〃	デバイスナンバーセット信号
TSPF3	4AH	:Indicator ランプ
	BIT7 〃	ERR Indicate
	6-0 〃	DATA 〃
FDCST	OE4H	:FDC(OPD765)STATUS REG(入力のみ)
	BIT7 〃	Request for Mastor
	〃 6 〃	DATA INPUT(1)/OUTPUT(0)
	〃 5 〃	NON DMA モード
	〃 4 〃	FDC BUSY
	〃 3-0 〃	FDD#3-#0 BUSY
FDCRW	0E5H	:IATA(レジスタ)READ/WRITE ポート
FDCNT	OE6H	:FDD コントロールポート
	BIT7 〃	Reset(1)信号出力;INT 信号入力
	〃 6 〃	モータON(1)信号出力
	5-4 〃	PRE-SHIFT I/O コントロール(DON.T USE=00)
	3-0 〃	
FDCT0	0E2H	:FDC TC 信号 ON
FDCTF	0E0H	:FDC TC 信号 OFF
DBCNS	0F0H	:DEBUG console インタフェイス

H. MINI-PASCAL 内部ルーチン一覧表

アドレス	内　容
0249	Accに画面モード(0～2)を入れてコールすると，モード設定，画面クリアを行う．
0450	カーソルをブリンク表示．
045E	カーソルを消す．
05E8	カーソル位置を移動する． Bレジスタ，CレジスタにX，Y座標を入れる．
065F	座標のドット情報を求める． FE5E，FにX座標，FE60にY座標をセットする． Accは，ファイングラフィックモードでドットがある，ない，グラフィックモードの3つに応じて，0，1，2となり，Dレジスタは，色コードが入ってくる．
05AA	HLレジスタペアで指定されたアドレス以降のデータをカーソル位置から，BCレジスタペアの個数分だけ表示する．
0606	FE5E，FにX座標，FE60にY座標，Accに色コードを入れてコールすると，ドットがセットされる．
061F	上と同様に座標を入れてコールすると，ドットがリセットされる．
0E3F	Accに押したキーのコードが入る．何も押されていないと，zeroフラグがonになる．
0EAF	Accの文字コードを出力．
2342	ミニフロッピーディスクのアクセス．FEABにコマンドコードを入れてコールする． **内容** / **コード(16進数)** Seek & Read — 86 Seek & Write with check — C5 Seek & Write IP — 8D Return to Zero — 07 Sense Drive Status — 04 Seek & Write — 85 Read Data — 06 Write Data — 05 Seek — 0F Write ID — 0D

	FEAC……FD1～4に応じて00～03. FEAD, E……データの先頭アドレス FEAF……データ長－1 FEB1……シリンダアドレス FEB2……セクタアドレス FEB3……ヘッド番号 FEB4……リトライ回数 結果は，アスキーコードでFE3Cに入る. 0： FDD NOT READY 2： ID FIELD ERROR 3： SEEK ERROR & DEFFECTIVE TRK検出 4： CRC ERROR 5： WRITE & CHECK ERROR & FILE USAFE ERROR 6： DELETED SECTOR 検出 7： 入力パラメータエラー 8： 正常終了 9： DELETED SECTOR での CRC ERROR G： WRITE PROTECT ERROR
3731	CLOAD コマンド 7CE1～ ファイル名
3801	BEGIN コマンド
381B	END コマンド
383F	JUMP コマンド
385C	FIND コマンド
38EE	<コマンド
397E	>コマンド
3A40	INSERT コマンド(7C54, 5…行番号)
3B20	LIST コマンド(7C54, 5, 6, 7…行番号)
3BC7	LIST#コマンド(〃)
3C8E	EDIT コマンド(7C54, 5…行番号)
3D67	DELETE コマンド(7C54, 5, 6, 7…行番号)
4524	プロンプトを表示して，1文字入力があるまで待つ. アスキーコードが Acc に入る.

45BE	8バイトのアスキーコードを，6バイトに変換する． 7D62～9：文字列 7C58：文字列長 HLレジスタペア：出力エリアのアドレス．
45E8	上の逆の操作を行う． 出力アドレスは，文字列が7DE3～，文字列長7C54．
4609	HLレジスタペアのアドレスから，Accのバイト数だけの数字を表す文字列を，DEレジスタペアに入れる．(バイナリデータへの変換)
466F	数字を表す文字列が#ではじまる16進数の場合に上と同じはたらきをする．
46DB	4609の逆のはたらきをする．7C54，5のバイナリデータを，7DE3からに，数字列に変換して入れる．そのケタ数は7C56に入る．
4760	466Fの逆のはたらきをする．入出力アドレスは46DBに同じ．
4AB9	SIZE コマンド
5BE0	FOR文(7C4C，D…アドレス)
5CCE	REPEAT 文(　〃　)
5D0C	WHILE 文(　〃　)
5D54	IF 文(　〃　)
5E71	関数，手続の実行(　〃　)
63BF	標準手続の実行 DEレジスタペア　第1パラメータアドレス Bレジスタ　パラメータ数 HLレジスタペア　手続のロケーション(以下のもの) CHR ――63C6 INSERT ――63E9 DELETE ――645C READ ――64BE WRITE ――6637 POKE ――6710 OUT ――672D MOVE ――675E CALL ――6844 COLOR ――7843 LINE ――7889 PSET ――79C8 PRESET ――7A1B SCREEN ――7A4D WIDTH ――7A73

6 8 9 C	GOTO X Y(DEレジスタペア，Bレジスタは同上)
6 8 D B	GETKEY(　　〃　　)
6 9 1 C	TIME をHLレジスタペアに入れる．
6 9 8 3	標準関数の実行 DEレジスタペア：第1パラメータアドレス Bレジスタ：パラメータ数 HLレジスタペア：関数のロケーション(以下のもの) ORD ——698A LENGTH ——69A5 DEEK ——69BB INP ——69CD ADR ——69F4 POINT ——7ABD 出力は， HLレジスタペア：関数の結果 Bレジスタ：データの型
6 A 8 1	データ比較 HLレジスタペア：比べられるデータのアドレス DEレジスタペア：比べるデータのアドレス Bレジスタ：判定コード Cレジスタ：データ型(0—整数形，1—文字型) 出力は，HLレジスタペア ｝0—偽 Bレジスタ ｝1—真 判定コード 52——= 53——< 54——> 55——<= 56——>= 57——<>
6 C B 1	2バイトの加算(DE+HL=HL)
6 C B B	2バイトの減算(DE−HL=HL)
6 C C C	2バイトのOR(DE OR HL=HL)
6 C D 6	2バイトのXOR(DE XOR HL=HL)
6 C E 0	2バイトの乗算(HL＊DE=DEHL)
6 C E E	2バイトのAND(HL AND DE=HL)

6CF8	2バイトの除算(DE／HL＝HL…BC)
6D0B	2バイトのMOD(DE　MOD　HL＝HL)
75ED	LOADコマンド
765D	SAVEコマンド
7731	PURGE オマンド
7750	CLIST コマンド
775F	CATALOG コマンド
77AB	エラーメッセージ出力 Acc：エラーコード コード　　メッセージ 35　NOT　FD1～FD4 36　NOT　PASSWORD 37　NOT　CRT　MODE 38　NOT　CRT　SIZE 39　DIRECTRY　FULL 40　FILE　OVER 41　FDD　OVERFLOW 42　FILE　NOT　FOUND 43　NOT　MINI　PASCAL 44　FDD　HARD　ERR

索 引

TECHNICAL BANK

本書の内容抜粋

ハードウェア仕様
チップ構成
割込機能
ディスプレイ装置とそのインタフェイス
ディスクのデータ転送とタイマの補正
ROM/RAMPAC
RS-232Cインタフェィス詳解
RS-232Cのデータ転送のタイミング
T-BASICのメモリ内部の状態
内部ルーチン・ポインタを使う
RAMを32K増やす(未使用RAMの活用)
メモリをALL-RAMに
プログラムの回復法
1つのキャラクタを複数色で
アトリビュート・キャラクタで高速グラフィックを
スクリーンモード1.5(160×200フルカラーモード)
ディスクへの直接書き込み
プリンタIIの逆スクロール機能を使う
リアルタイムキースキャン
漢字パターンと漢字ファイル
OA-BASICのメモリ内部の状態
インデクス・シーケンシャル・ファイルについて
PASCAL(パスカル)とは?
PASCALの再帰的呼び出し
MINI-PASCAL内部の状態
T-BASICとMINI-PASCALの総合的比較
CP/Mとは?
CP/Mコマンドの実行のされ方

パソピアの内部構造

定価2,800円

ISBN4-87148-700-8 C3055 ¥2800E